Performance Analysis of Multicomputer Interconnection Network Designs

by

Hassan Z. Abdalla

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

APPROVED:

S. F. Midkiff, Chairman

J. R. Armstrong

C. E. Nunnally

November, 1987

Blacksburg, Virginia

Performance Analysis of Multicomputer Interconnection Network Designs

by

Hassan Z. Abdalla

S. F. Midkiff, Chairman

Electrical Engineering

(ABSTRACT)

In this thesis, the design and performance issues of multicomputer interconnection networks are addressed. Analytic models are used to evaluate the performance of large scale multicomputer networks. Performance is measured as the average end-to-end delay between communicating nodes. The models consider the communication processing queueing delays experienced by packets at each node as well as delays associated with transmission of packets. A comparison study of message switching and cut-through switching is presented. It is shown, conforming to previous studies, that a network has superior performance when cut-through switching is used. Cut-through switching is most advantageous when the network load is low and degenerates to message switching performance under heavy network loads.

The network model is used to develop a "Network Analyzer", an interactive program that allows analysis of different network designs and work loads. Torus and spanning bus hypercube networks are then analyzed by the program to study the effect of various network parameters, traffic patterns, and switching techniques on network delay performance. The Network Analyzer is relatively easy to learn and use, and once learned, it is easy to change network parameters to perform a trade-off study of design alternatives. The program can also be used to quickly generate listings and graphics to illustrate the results of the analysis. The program proved to be an efficient design tool.

Hypercube topologies are the most popular for large scale parallel computers, particularly for computationally intensive applications. The torus and spanning bus hypercubes are implemented in our program and analyzed in detail. Results show that network topology has a mixed impact on system performance. Networks built of standard modules of spanning bus hypercubes are harder to expand, compared to a torus network, but they have lower connection cost per node. Communication delays are generally higher for spanning bus networks except at very low traffic loads.

Acknowledgements

I sincerely thank everyone who has contributed towards the completion of this thesis. I especially thank Dr. S.F. Midkiff, my principal advisor, for the countless number of hours he spent offering guidance, information, and suggestions. I thank the other members of my graduate committee, Dr. C.E. Nunnally and Dr. J.R. Armstrong, for their comments and help.

With great appreciation, I thank my wife for the support, encouragement, and love necessary for me to complete this work.

Table of Contents

List of Illustrations

List of Tables

1.0 INTRODUCTION

Recent advances in VLSI have made large multicomputer networks a promising approach to parallel processing. Multicomputer networks, with point-to-point links between processing nodes, use processing nodes operating concurrently to solve problems decomposed such that each node executes a small part of the problem. Each node contains a functional processor that executes computational tasks and a communication processor that executes communication tasks. Since the processing nodes need to share information, the multicomputer interconnection network must efficiently support message transfer. In many multicomputer systems it is the communication time rather than the computation time that limits system performance [1]. Optimization of message transfer time, through proper design of interconnection networks and system traffic patterns, are thus crucial. The objective of this research is to evaluate the performance of multicomputer network alternatives and to develop a design aid that allows an accurate and efficient analysis.

Analytic models of large scale multicomputer networks can be used to determine end-to-end delays between the nodes and overall system throughput. They provide a framework for testing alternative node designs, traffic patterns, and network topologies.

In this research existing models will be generalized and extended to reflect different switching techniques and traffic patterns. A comparison study of store-and-forward packet switching and cut-through switching is presented. The comparison is based on network delay performance. Several static network topologies are analyzed to determine their mean internode distances as well as their link traffic loads under different traffic patterns. Analytic models are then used to develop an analysis tool, an interactive program that allows analysis and comparison of different network designs and work loads. Appropriate performance evaluation parameters are selected from the following options:

- Traffic patterns
- Network topologies and size
- Switching techniques
- Link transfer rates (bandwidth)
- Message length
- Header length
- Processing time

Throughout our research we make the following general assumptions:

1. Composite traffic arrival at each node is Poisson distributed.

2. Message lengths, and therefore message transmission times, are exponentially distributed.

3. Message queues have infinite nodal storage capacity.

4. Network topologies are symmetric.

5. Nodes are identical, i.e. all communication processors and links are the same.

6. The average message generation rate is uniform across all nodes.

7. Given assumptions 4, 5, and 6 above, traffic patterns cause uniform load conditions, i.e., the patterns are such that the message rates over all links are the same.

Assumptions 1, 2, and 6 allow analysis to be tractable and are reasonable for large networks with varying message sizes and types. Infinite nodal storage capacity is a reasonable assumption for properly designed communication processors and keeps analysis from being bogged down in complexity. Assumptions 4 and 5 are true for many popular network topologies. Although a network topology can be arbitrarily designed, regular and symmetric topologies are frequently used for ease of implementation. The last assumption follows from assumptions 4, 5, and 6.

Network design issues are discussed in the next chapter. Design decisions are made between network topologies, switching methods, operation modes, and control strategies. Performance evaluation measures are the subject of Chapter 3. Network geometry characteristics and message traffic characteristics are discussed. Network delay models are developed to allow analysis under different network topologies, traffic patterns, and packet generation rates. Static performance measures for two network topologies, the torus and the spanning bus hypercube are then presented in Chapter 4. Chapter 5 describes the Network Analyzer program and provides a user guide. This is followed by application examples and conclusions in Chapter 6 and Chapter 7, respectively.

2.0 DESIGN ISSUES OF INTERCONNECTION NETWORKS

2.1 INTRODUCTION

In earlier times, when computer systems were confined within von Newman architecture and hardware cost was a limiting factor, interprocessor communication was not a prominent issue. Consequently, the design of cost effective interprocessor communication was an unimportant task. Today, the demand for very high speed processing coupled with falling hardware costs has made large-scale parallel and distributed computer systems both desirable and feasible. A number of parallel computer architectures, where several microcomputers are connected by an interconnection networks, have been proposed in response to the ever growing need for speeding up computationally intensive tasks [1-3]. Each node in these multicomputer networks includes a computation or functional processor with some local memory, a communication processor, or controller, that supports internode message transfer and processing without delaying the computa-

tion processor, and a small number of connections to other nodes. "An interconnection network consists of software and hardware entities that are designed to facilitate efficient interprocess and interprocessor communication" [4].

An objective of this research is to assess the performance of alternative interconnection network designs. As a starting point, network design decisions, as outlined in [5,6], are discussed below. These are fundamental decisions that determine the appropriate architecture of an interconnection network. The space of the interconnection networks can be represented by the cartesian product of the following four sets of design features: (operation mode) X (control strategy) X (switching methodology) X (network topology). Not every combination of the design features is interesting. The choice of a particular interconnection network depends on the application demands, technology support, and cost-effectiveness.

2.2 NETWORK TOPOLOGIES

Network topology is a key factor in determining a suitable architectural structure. A network can be depicted by a graph in which nodes represent switching points and edges represent communication links. The topologies tend to be regular and can be grouped into two categories: static and dynamic [6]. In a static topology, links between two processors are passive and dedicated buses cannot be reconfigured for direct connections to other processors. On the other hand, links in the dynamic category can be reconfigured by setting the network's active switching elements.

2.2.1 Overview of Static Networks

Topologies in static networks can be classified according to the dimensions required for layout. For illustration, one-dimensional, two-dimensional, and three-dimensional networks are shown in Figure 1. Examples of one-dimensional topologies include the linear array used for some pipeline architectures, shown in Figure 1-a. Two-dimensional topologies include the ring, star, tree, mesh, and hexagonal array, also called systolic array. Examples of these structures are shown in Figures 1-b through 1-f. Three-dimensional topologies include the completely connected chordal ring, 3-cube, and 3-cube-connected-cycle networks depicted in Figures 1-g through 1-j. A D-dimensional, W-wide hypercube contains W nodes in each dimension, and there is a connection to a node in each dimension. The mesh and the 3-cube are actually two- and three-dimensional hypercubes, respectively. The cube-connected-cycle is a deviation of the hypercube. For example, the 3-cube-connected-cycle shown in Figure 1-j is obtained from the 3-cube.

In a global bus topology [2], all nodes are directly connected to a common bus. A message from any node propagates through the bus and can be received by all other nodes. Because all nodes share a common transmission link, only one node can transmit at a time. Simultaneous requests for bus access are resolved by some form of contention resolution protocol. A global bus system can support only a small number of nodes because message density increases linearly with the total number of nodes in the network [2]. In contrast, the completely connected networks [7] have a dedicated link between each pair of nodes. This eliminates completely link access contention but the number of connections per node grows quadratically with the number of network nodes. These

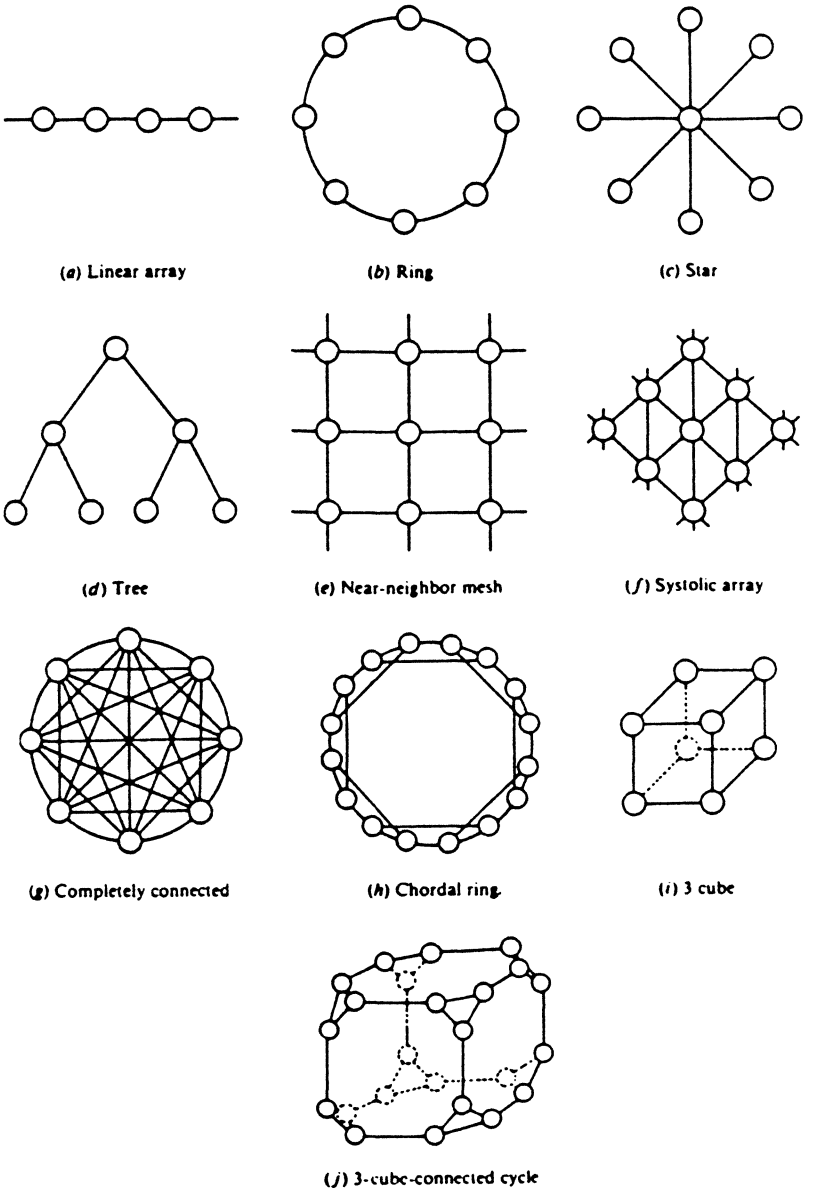


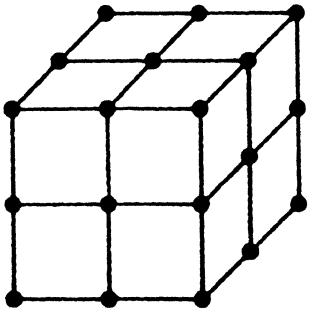
Figure 1. Static Interconnection Network Topologies.: (From Feng [5]).

two networks, the global bus and the completely connected, bound the spectrum of cost and performance for all practical multicomputer networks [8].

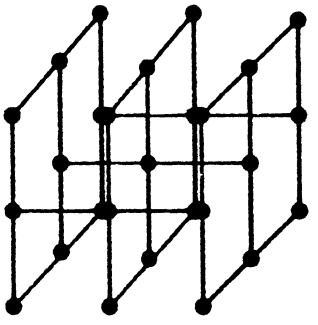
In a ring topology, nodes form a closed loop and messages are relayed from node to node around the loop. The ring topology can support a limited number of nodes because average message delay and message traffic density increases linearly with the number of nodes in the ring [2]. The chordal ring [9] is a ring structured network in which each node has an additional link, called a cord, to some other node across the network. This reduces the number of links that must be travelled to reach a destination node.

2.2.2 Hypercube Connection Structures

The hypercube topology is the most popular architecture for large-scale parallel computers, particularly for computationally intensive applications. One attraction of this architecture is that its flexible communication network let programmers choose different topologies for different applications [10]. For example, a four-dimensional system, with 16 nodes, can be treated as a two-dimensional mesh, a three-dimensional mesh, a ring, or a tree merely by directing communication appropriately between nodes [10]. Several connection structures have appeared in the research literature for the design and analysis of hypercube interconnection networks. "Even though connection costs are well controlled, these structures can allow message delays to increase as slowly as log N, where N is the total number of nodes in the network, so that a large number of computers may efficiently communicate with each other" [2]. Among the proposed structures are the spanning bus hypercube [2], the dual-bus hypercube [2], the torus [2], generalized hypercube [3], and the cube-connected cycles [11]. The spanning bus



(a) 3-Dimension spanning bus hybercube



(b) 3-Dimension dual bus hybercube

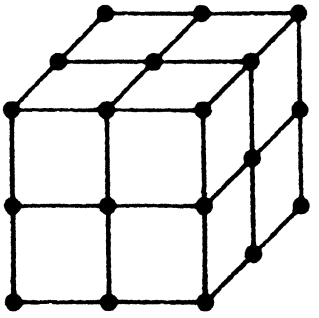
Figure 2. Dual bus versus spanning bus hybercube structures.

hybercube is a D-dimensional lattice of width W in each dimension. Every node is connected to D buses, each bus spanning a different dimension in the hypercube space. W nodes share a bus in each dimension.

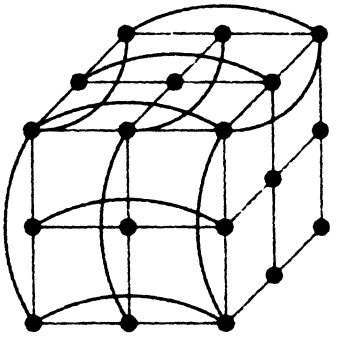
The dual-bus hypercube was proposed to limit the number of connections to each node. In this structure each node is connected to two buses only. One dimension, the 0th dimension, is distinguished, and all nodes are connected to a 0th-dimension bus, shown as the vertical direction in Figure 2. In each $D - 1$ hyperplane perpendicular to the 0th dimension, all nodes have their second connection to buses spanning the same dimension. The second bus direction differs from plane to plane but may repeat if the width W of a dimension exceeds $D - 1$. The torus, Figure 3, is a W^D hypercube with end-around connections. It is identical to the spanning bus hypercube except that the bus connecting each group of W nodes is replaced by a ring of point-to-point connections. Generalized hypercube structures use a mixed radix number of nodes, in contrast to traditional hypercube structures where $N = W^D$ for some integer values of W and D.

The cube-connected cycle topology was also proposed to limit the number of connections per node. This topology is a deviation of the hypercube. For example, the 3-cube-connected cycle shown in Figure 1-j is obtained by replacing each node of the 3-cube by a 3-node cycle. Each node in the cycle is connected to the corresponding node in another cycle.

Each of these networks can be analyzed to determine its mean internode distance and its traffic density, two key factors that play a critical role in determining network delay, as will be shown later. The torus and spanning bus hypercubes are analyzed in detail in Chapter 3 and are implemented in the Network Analyzer program.



(a) 3-Dimension spanning bus hybercube



(b) 3-Dimension torus

Figure 3. Torus versus spanning bus hybercube structures.

2.3 SWITCHING TECHNIQUE

The two major switching methodologies are circuit switching and packet switching. In circuit switching, a physical path is established between a source and a destination. In packet switching, data is formatted as a packet and routed through the interconnection network without establishing a complete physical connection path. In general, circuit switching is more suitable for bulk data transmission, and packet switching is more efficient for many short data messages. Another option, integrated or virtual cut-through switching, includes the capabilities of both circuit switching and packet switching. Therefore, three switching methodologies can be identified: circuit switching, packet switching and virtual cut-through switching.

2.3.1 Circuit Switching

With circuit switching, a complete path of communication links must be set up between two communicating nodes before the real communication begins. This is accomplished by a signalling message. The path is tied up during the entire session between the two nodes. Once a path is set up, no further signalling for addressing purposes is necessary. Thus, in a circuit switched network, a path, once set up, implicitly provides all the addressing information. When applied to data communication networks, circuit switching suffers from some drawbacks [12]. One is the slow set up which delays transfer of messages from sender to receiver. Another drawback is the low channel utilization due to the fact that the channels on a path are tied up, but are not actually being used during idle periods. That is, the dynamic assignment of paths is not dynamic enough.

Figure 4 shows the network delay in different switching systems. In this figure, it is assumed that there is no interfering traffic and that the number of intermediate nodes in the path is two. It is also assumed that message processing time is negligible.

2.3.2 Packet Switching

In order to achieve a better channel utilization, one may think of relinquishing the channels on a path during periods in which the source and destination are not communicating. This brings us to the idea of "store-and-forward" packet switching. Packet switching is loosely used here to refer to any store-and-forward protocol in which data are copied into holding buffers at each intermediate node along some path from source to destination. No assumption is made about packet size; packets may conceivably be large enough to encompass any single message.

In this method messages are routed toward their destination node without establishing a path beforehand, rather, the paths are assigned dynamically. Through provision of a storage facility at each node, messages are stored in intermediate nodes and then are sent forward to a selected adjacent node, hence the name store-and-forward. This process is repeated until the message reaches the destination node. By attaching addressing bits to the header, each message carries information regarding its destination. Since communication links are not allocated into complete paths for specific source-destination pairs, each link is statistically shared by many nodes. Figure 4-b shows the network delay in a message switching system.

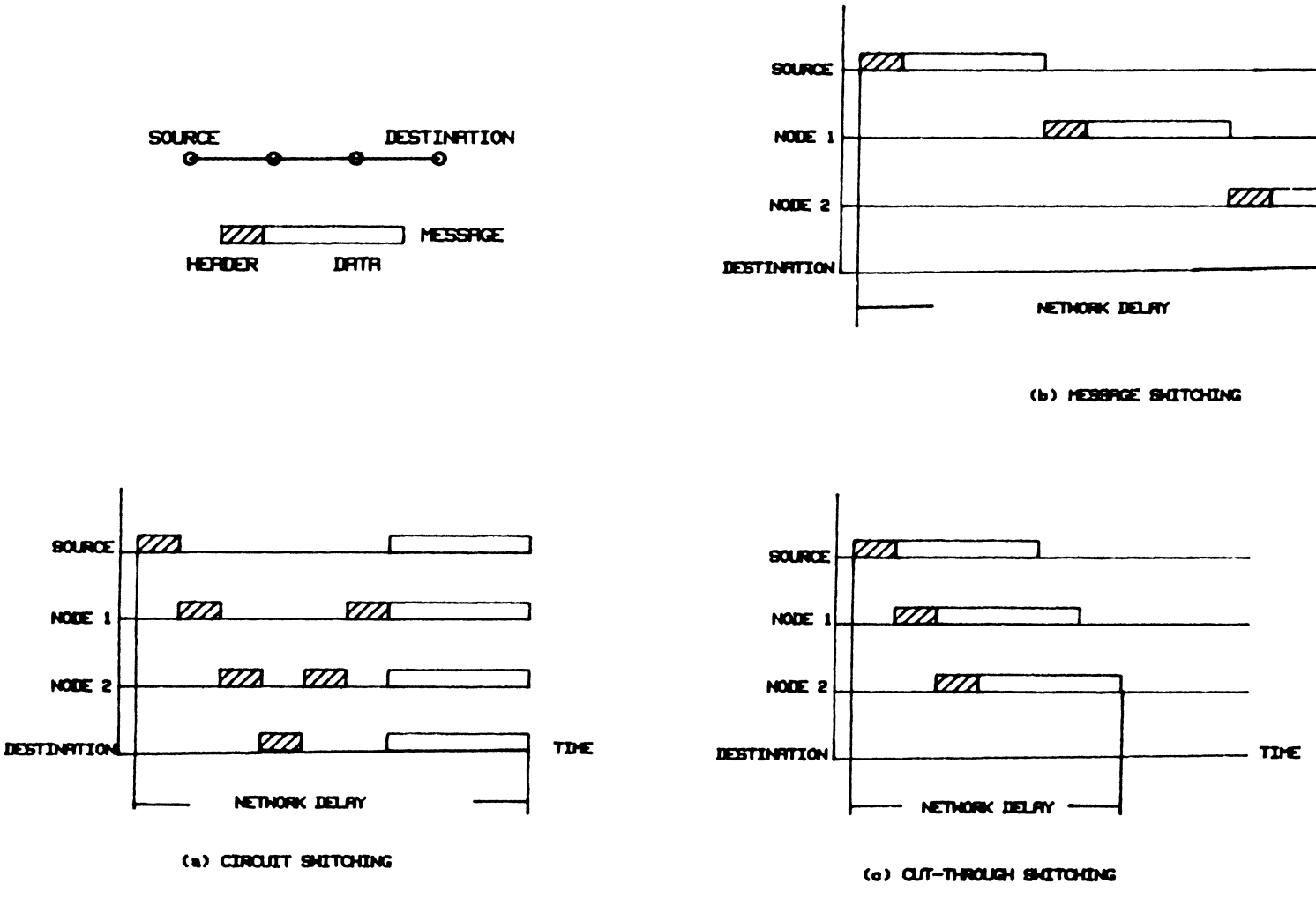


Figure 4. Network delays for different switching techniques.

2.3.3 Virtual Cut-Through Switching

From Figure 3-c we observe that extra delay is incurred because a packet is not transmitted out of a node before it is received completely. When a message arrives in an intermediate node and its outgoing channel is free, it actually does not need to be completely received in the node before being transmitted out. In the virtual-cut-through protocol [13], intermediate nodes along a message path attempt to send messages onward as soon as an appropriate output link has been determined. If the appropriate output channel is free, the attempt succeeds; output and input continues in parallel, with the initial portion of the message being transmitted while the final portion is being received. Otherwise, the message is accepted and buffered as per normal store-and-forward procedure. Virtual-cut-through, thus, attempts to pipeline a message through the network at a grain size determined by the time required for routing at each intermediate node. If the packet encounters busy channels at all of the intermediate nodes, the outcome is exactly the same as in a packet switched network. On the other hand, if all the intermediate channels are free, the outcome is exactly the same as in a circuit switched network without the overhead of initial signalling. Cut-through is most advantageous when the network load is low and degenerates to packet switching performance under heavy network loads.

2.4 OPERATION MODE

Two types of communication can be identified: Synchronous and asynchronous [5]. Synchronous communication is needed for either a data manipulation function or for a

data instruction broadcast. Asynchronous communication is needed for multiprocessing in which interprocessor connection requests are issued dynamically. A system may also be designed to facilitate both synchronous and asynchronous processing. Therefore, the typical operation modes of interconnection networks can be classified into three categories: synchronous, asynchronous, and combined. In our research, no assumption is made regarding operation mode because it will have no bearing on either network or node models.

2.5 CONTROL STRATEGY

A typical interconnection network consists of a number of switching elements and interconnecting links. Interconnection functions are realized by proper setting of the switching elements. The control-setting function can be managed by a centralized controller or by the individual switching element. The latter strategy is called distributed control and the first strategy corresponds to centralized control.

In circuit switching networks, a circuit path has to be established before data is actually transmitted over the path. Therefore, the control or routing problem must be solved in the path connection phase by specifying setting of switching points to be used in the path. The switching setting is usually derived from source and destination addresses. The switching points are set to a specific connection state by a centralized controller, in the case of centralized control, or by itself in the distributed control case. In packet switching networks, each data packet is routed through the network according to the routing message incorporated in the data packet. The distributed control ap-

proach is usually used in packet switching networks. This agrees with our assumption, through out this research, of identical node structure.

"Networks with static topologies usually use distributed packet routing" [4]. A data packet contains a header with one or more routing parameters; the destination address is one of these parameters. The distributed control algorithm is implemented in individual communication nodes, which decode the routing parameters upon reception. The node then determines the next node to route the data packet to and possibly updates some of the routing parameters to reflect data routing history. A typical routing strategy can be found in [9] for chordal ring networks. In Section 4.3, we give an example of a packet header and how the routing parameters can be updated. Conflict resolution and queueing are needed when there are multiple inputs and/or multiple outputs at a communication node. In our model, nodes have multiple input and outputs. Communication queues are served on first come-first serve basis.

3.0 PERFORMANCE EVALUATION

3.1 INTRODUCTION

Performance evaluation measures network capability in terms of parameters that represent major characteristics of a network application model. Metrics used include average message delay, message density per link, bandwidth, throughput, effectiveness in simulating other networks, acceptance rate for connection requests, expected capacity, number of permutaions that can be performed, number of crosspoints needed, average number of routing passes or steps for a data communication, degree, diameter, area-time product of a VLSI realization, reliability, degree of fault tolerance, number of redundant paths, and cost. Some of the performance metrics are related to inherent geometric characteristics of the networks while others are related to traffic conditions resulting from communication requests.

Performance evaluation research on networks with static topologies [2], [8], [14], is limited, relatively, compared to the enormous work performed on the evaluation of networks with dynamic topologies [15-23]. Message delay and message density of some

static networks such as loops, cubes, and trees are derived under an assumption of uniform distribution of messages across the networks [2]. The throughput of a looped static baseline network is assessed [14].

For networks with dynamic topologies, some evaluations are done on parameters such as number of crosspoints [15], effectiveness in simulating other networks [16], [17], and combinatorial power [18], which are not related to traffic. The majority of work is concerned with traffic-related parameters such as bandwidth and throughput. Both circuit switching [18-20] and packet switching [21-23] are considered. Analytic methods, such as queueing theory and Markov chain theory, and simulation methods are used. Simplified models have been used in the analysis. Most of the models assume random and uniform requests from processors.

In our study we measure network performance in terms of end-to-end communication delay. Generally, traditional computer models are not applicable to multicomputer systems, except for derivation of order of magnitude performance estimates under very simplifying assumptions. Instead such systems must be treated as "network computers" [26]. A network computer is a network of processor nodes that is intended to function not as a collection of autonomous hosts but as a single MIMD machine. "Network computers are designed to support asynchronous distributed programs" [24]. As the cost of multiprocessors falls, such machines become increasingly affordable and interest has increased in their performance evaluation. Application of classic network models [25] is inappropriate due to significant parameter deviations from ordinary point-to-point computer communication networks. In computer communication networks, communication processing queueing delays are, in general, very small compared to the associated transmission delays. Consequently the classic network model takes into consideration only outgoing link queues. However, the communication links em-

ployed in multicomputer systems have very high bandwidths and thus transmission delays no longer play a dominant role.

The model considered here is adapted from [26] and provides a realistic, and therefore more accurate model of a network computer. The original model of [26] assumes a packet switching technique and point-to-point links. The communication queueing processing delays experienced by a message or a packet at each node along its route to the destination is explicitly factored into the model. As it turns out, communication processing delays play a predominant role in the determination of the performance of network computer systems. Network traffic thresholds are determined by communication processing queueing rather than transmission delays and are generally lower than those predicated by a classic network model. In the following, the model is reviewed and generalized to reflect the effect of different link traffic densities. The model is then modified to permit analysis of the virtual cut-through switching technique.

3.2 DEFINITIONS AND BACKGROUND

Before developing the model, it is useful to define several important terms and concepts used to quantify the performance of networks as outlined in [8].

3.2.1 Network Geometry Characteristics

A network topology dictates the network geometry characteristics, some of which can affect average message delay through the network. Three important network char-

acteristics are defined below, network diameter, mean internode distance, and network symmetry. Note that the mean internode distance of a network depends also on the message routing distribution through the network.

Network Diameter: The maximum internode distance, often referred to as the diameter of the interconnection network, places an upper bound on the delay required to propagate information throughout the network. It is simply the maximum number of communication links that must be traversed to transmit a message between any source-destination node pair along a shortest path.

Mean Internode Distance: In contrast to the network diameter, the mean internode distance is the expected number of link traversals a typical message makes to reach its destination. The mean internode distance is a better indicator of average message delay than the network diameter. Unlike the network diameter, the mean internode distance depends on the message destination distribution. This destination distribution specifies the probability that different node pairs exchange messages, and it ultimately depends on the communication requirements of the application and system programs as well as on the mapping of these programs onto the network. In its most general form, the mean internode distance is given by [8]

$$N_h = \sum_{l=1}^{lmax} l \times \varphi(l)$$

where $\varphi(l)$ is the probability of an arbitrary message crossing l communication links, i.e., the routing distribution, and $lmax$ is the network diameter. Different choices for $\varphi(l)$ lead to different message destination distributions and, in turn, different mean internode distances. In the following, we consider two different message distributions for which

it is possible to obtain closed forms for the mean internode distance. To do this, however, we must first distinguish between two types of interconnection network topologies: symmetric and asymmetric.

Network Symmetry: "In a symmetric interconnection network there exists an isomorphism that maps any node of the network graph onto any other node" [8]. Thus, all nodes possess the same view of the network. A bidirectional ring network is a simple example of a symmetric interconnection network because two nodes are always reachable by crossing any given number of communication links, and a simple node renumbering suffices to map any node onto any other. An asymmetric interconnection network is any network that is not symmetric, e.g., a tree. Although the network topology can be arbitrarily designed, regular and symmetric topologies are frequently used to ease the physical implementation task. In this research, we consider only symmetric networks.

3.2.2 Traffic Characteristics

Traffic patterns are characterized by the way messages move between communicating nodes in a multicomputer network. In the following, two traffic patterns are distinguished, uniform traffic pattern and sphere of locality traffic pattern. The notion of "balanced networks" is also discussed.

Uniform Message routing: A message destination distribution is said to be uniform if the probability of sending a message from one node to another is the same for all nodes in the network. Because we are interested in message transfers that use the network, we

exclude the case of nodes sending messages to themselves. Uniform routing distribution is appealing because it makes no assumptions about the type of computation generating the messages; this is also its greatest liability. Because most computations should exhibit some measure of communication locality, it provides what is likely to be an upper bound on the mean internode message distance.

Sphere of Locality: Suppose the uniform message routing assumption is relaxed. One would expect any reasonable mapping of a distributed computation onto a multicomputer network to place those tasks that exchange messages with high frequency in close physical proximity. One abstraction of this idea places each node at the center of a sphere of locality with radius L, measured in hops. A node sends messages to the other nodes inside its sphere of locality with some, usually high, probability φ and to nodes outside the sphere with probability $(1 - \varphi)$. This model reflects the communication locality typical of many programs, e.g., the nearest-neighbor communication typical of iterative partial differential equation solvers coupled with global communication for convergence checking [8].

Balanced Networks: "A network is said to be balanced if the traffic across all its links are the same" [13]. This property does not imply any assumption about the topology of the network. In light of network symmetry assumption and the two message routing distributions defined above, networks considered here are balanced.

3.3 NETWORK MODEL DEVELOPMENT

A section of the overall network and the assumed structure of each node is depicted in Figure 5. The node includes a functional processor (FP) that executes computation tasks. A communication processor (CP), together with a number of communication controllers (CC), provide the means for communication with the functional processors of the other nodes. The original model of [26] assumes bidirectional internode links, i.e., links act as half-duplex channels. "This assumption can be relaxed so that we may have *D* full-duplex channels per node, each channel consisting of two unidirectional links" [27].

Figure 6 shows the queueing model for a single node adapted from [26]. In this model, arriving messages are stored by the CC's in shared memory for subsequent processing by the CP. Arriving messages, together with messages generated by the FP in that particular node, form a queue in front of the CP. The CP accesses the header part of each message and executes a routing algorithm to determine the message destination. It then forwards the appropriate commands to either the resident FP or the appropriate CC depending on the message's final destination. Message arrivals at a CP are assumed to follow a Poisson distribution while the time required to route each message is assumed to be fixed. It follows that this first stage can be modeled as an M/D/1 queuing system. The composite arrival rate, λ_{cp}, is given by [26]

$$\lambda_{cp} = \lambda_{fp} + \sum_{i=1}^{D} \lambda_{l,i}$$

where λ_{fp} denotes the mean message generation rate by the resident FP and $\lambda_{l,i}$ denotes the mean message arrival rate over link *i*.

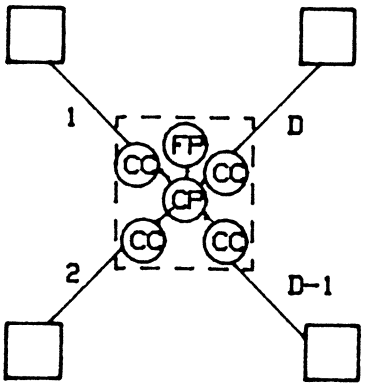


Figure 5. Multicomputer system node structure.

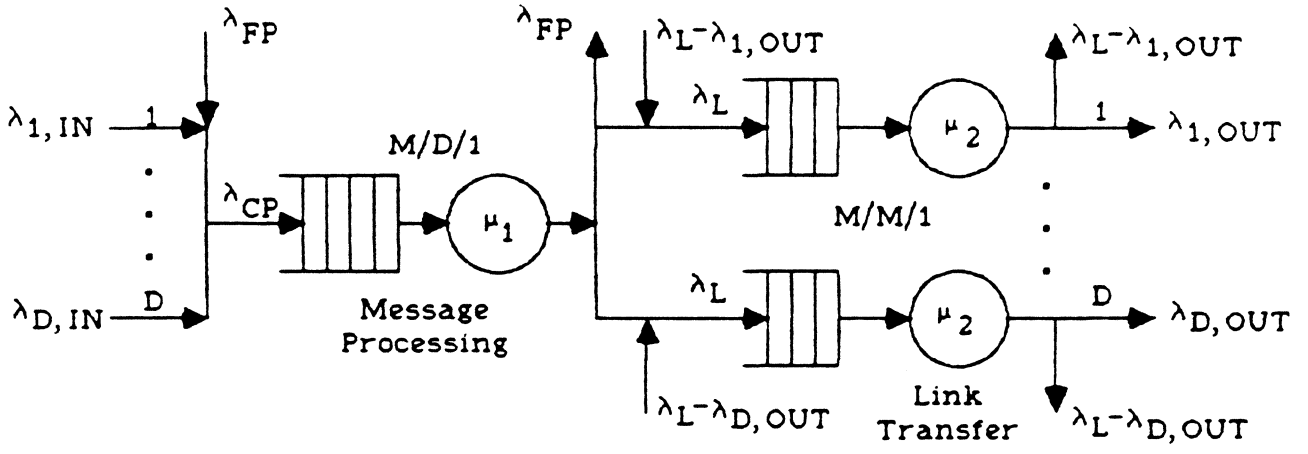


Figure 6. Queueing model of a network node.

The total processing time for the first stage, T_{cp}, at each node, which includes waiting time and service time, is thus obtained as

$$T_{cp} = \frac{1}{\mu_1} + \frac{\lambda_{cp}}{2\mu_1(\mu_1 - \lambda_{cp})} \tag{3.1}$$

where $\frac{1}{\mu_1}$ is the CP fixed processing time [26].

"Even though interdepartures from the M/D/1 stage are generally non-Poissonian, interarrivals at each link queue can be approximated very closely with a Poisson distribution" [26]. If message lengths are assumed to be exponentially distributed then their associated transmission times are also exponential and link queues can be modeled as M/M/1 queues. Properties of the M/M/1 queue are described in Appendix B. For a balanced network, defined earlier, traffic going from the M/D/1 queue to any link, or equivalently to any M/M/1 queue, is equally likely. Thus the composite arrival arrival message arrival rate at each of the M/M/1 queues is the same and the total delay through this second stage, T_l, is given by

$$T_l = \frac{1}{\mu_2 - \lambda_l} \tag{3.2}$$

where $\frac{1}{\mu_2}$ is the average message transmission time and λ_l is the aggregate traffic on one link. The total delay experienced by a message at a node, T_n, is the sum of the delay at the M/D/1 stage and the delay at the corresponding M/M/1 stage, i.e.,

$$\begin{aligned} T_n &= T_{cp} + T_l \\ &= [\frac{1}{\mu_1} + \frac{\lambda_{cp}}{2\mu_1(\mu_1 - \lambda_{cp})}] + \frac{1}{\mu_2 - \lambda_l} \end{aligned} \tag{3.3}$$

The overall message delay through the network will depend on the implemented switching technique and will be discussed in the following sections.

3.4 MESSAGE SWITCHING DELAY

In a network computer, processes running in different nodes exchange information through messages flowing via virtual channels. When nodes are not adjacent these virtual channels span through intermediate nodes. To simplify the analysis, we consider a specific communication path from source S to destination D, as depicted in Figure 7, and study the message delay along this path under different switching techniques. In this path model, there are $N_h + 1$ nodes connected in tandem. Messages enter the network at node 1 and leave the network at node $N_h + 1$, i.e., they travel N_h hops. Note that such a path model is quite general and applicable to almost any network topology.

In message switching, messages are transmitted in a hop-by-hop fashion through the network. Each message carries its destination address in its header. At each intermediate node the message must be completely received before it can be forwarded towards its destination node. The overall delay experienced by a message at a node is the sum of the delay at the M/D/1 stage and the delay at the corresponding M/M/1 stage. Considering that there are $N_h + 1$ message processing queueing stages and N_h link queueing stages along a route of N_h hops, then the mean end-to-end message transfer delays, T_{MS}, is

$$T_{MS} = (N_h + 1)T_{cp} + N_h T_l$$

Substituting for T_{cp} and T_l from Equations 3.1 and 3.2 we get the total delay in terms of the model parameters as:

$$T_{MS} = (N_h + 1)[\frac{1}{\mu_1} + \frac{\lambda_{cp}}{2\mu_1(\mu_1 - \lambda_{cp})}] + \frac{N_h}{\mu_2 - \lambda_1} \tag{3.4}$$

3.5 CUT-THROUGH SWITCHING DELAY

In cut-through switching, the operation is similar to message switching with the difference that messages do not have to be received completely at an intermediate node before being transmitted out of the node toward the destination. After the header of a message is received, the outgoing channel can be selected, and if this selected channel is free, the message may start transmission out of the node immediately, even though its tail has not yet arrived at the node. If, however, after the reception of the header, it is found out that the outgoing channel is busy, the operation follows that of message switching, i.e., the message is received completely before being sent out through intermediate nodes.

The required behavior of virtual-cut-through switching may call for more complex CC's at each node to avoid waiting for and using the CP to decode the message address. One way to alleviate the processing required by each CC is to use a routing algorithm where a source node generates not only the final node destination address, but also the address of each link to be used through the virtual channel. Figure 8 shows a possible address control field where k indicates the number of remaining hops along the routes. l_k is the local link number to be used in the current node. Note that $k=0$ indicates that

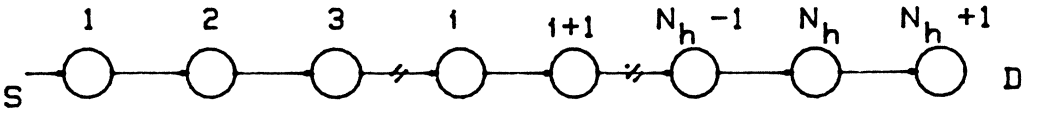
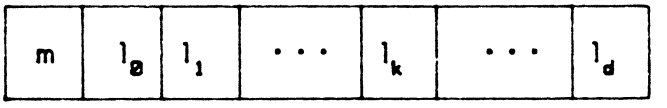


Figure 7. A typical communication path.



m = number of remaining hops

Figure 8. Example of a packet address control field.

the current node is the final destination. Whenever a message enters a free node it can make a cut. A node is called free if l_k is free. In this case, the message is passed immediately to the free link for transmission to the next node and k is decremented. If l_k is busy, then the message must be received completely before being transmitted out.

The probability of finding a free node, i.e., a free assigned link in a node, is the probability of having an idle M/M/1 queue. This event occurs with probability $(1 - \rho)$ [25], where ρ represents the link utilization factor and is defined as $\rho = \frac{\lambda_1}{\mu_2}$ (refer to Appendix B). Assuming that traffic on each link is independent, the number of cuts has a binomial distribution [13] and we have:

$$Pr(N_{cuts} = K) = \binom{N_h}{K}(1 - \rho)^K \rho^{N_h - K - 1}$$

and

$$\begin{aligned} N_c &= E(N_{cuts}) \\ &= \sum_{K=0}^{N_h - 1} KPr(N_{cuts} = K) \\ &= (N_h - 1)(1 - \rho) \qquad 0 \le K \le N_h - 1 \end{aligned} \tag{3.5}$$

where E denotes expectation and N_c denotes the mean number of cut-throughs. Note that the full message must be received at the $N_h + 1$ node, so at most $N_h - 1$ cuts can be made.

For each node at which a cut-through is made, a nodal service time, T_n less the header transfer time, is saved. However, this service time is conditioned on the event that the waiting time at link i is zero. Recall that a link is modelled as an M/M/1 stage.

Let T_{CT} be the mean end-to-end transfer delay using virtual cut-through switching. Ignoring the header transfer time, we have

$$T_{MS} - T_{CT} = N_c \; E(T_n | W_i = 0)$$

where T_n is the total delay in a node and W_i is the waiting at the ith M/M/1 stage.

Substituting for T_n from Equation 3.3 and noting that T_{cp} is independent of W_i we get

$$\begin{aligned} T_{MS} - T_{CT} &= N_c \, E((T_{cp} + T_l) | W_i = 0) \\ &= N_c \, [T_{cp} + E(T_l | W_i = 0)] \\ &= N_c \, [T_{cp} + \frac{1}{\mu_2}] \end{aligned}$$

Considering the fact that a message can be sent out only after its header is received, header transfer time can not be saved. The previous equation is changed to:

$$T_{MS} - T_{CT} = N_c \, [T_{cp} + \frac{1}{\mu_2} - \frac{\alpha}{\mu_2}]$$

where α is the ratio of header length to total average message length, including the header. Using the value of N_c from Equation 3.5, we get

$$T_{CT} = T_{MS} - (N_h - 1)(1 - \rho)[T_{cp} + \frac{(1 - \alpha)}{\mu_2}] \qquad (3.6)$$

Substituting for T_{MS} from Equation 3.3 we get

$$T_{CT} = (N_h + 1)T_{cp} + N_h T_l - (N_h - 1)(1 - \rho)[T_{cp} + \frac{(1 - \alpha)}{\mu_2}]$$

after some algebraic manipulations we obtain the following expression for T_{CT},

$$T_{CT} = [\rho(N_h - 1) + 2][\frac{1}{\mu_1} + \frac{\lambda_{cp}}{2\mu_1(\mu_1 - \lambda_{cp})}] + [\frac{N_h}{\mu_2 - \lambda_l} - \frac{(1 - \alpha)(1 - \rho)(N_h - 1)}{\mu_2}] \quad (3.7)$$

Two important cases can be recognized for the number of cut-throughs:

Case (1) $\rho = 1$. This implies $N_c = 0$, i.e., no cut-throughs are made and $T_{MS} = T_{CT}$

Case (2) $\rho = 0$. $N_c = N_h - 1$ and message switching offers the greatest saving in processing and transmission delays.

In case 1, with probability one, all of the intermediate nodes are busy upon arrival of the message and no reduction in delay is made. Under heavy traffic loads, messages are received completely and processed, at each node along its destination, and the network behaves like a pure message switched system. In case 2, with probability one, all of the intermediate nodes are free and all cut-throughs are made, thus the behavior resembles a circuit switching system. The message is stored and processed only at the source and destination nodes, consequently message buffering and processing times at the intermediate nodes are completely saved. Between the two extremities of link utilization factor, ρ , network delay is smaller for cut-through switching at low traffic loads, or small ρ, and degenerates gradually as traffic load grows. Equation 3.5 shows that the average num-

ber of cut-throughs increases as the mean internode distance, or equivalently the mean number of of hops, increases. This is intuitively expected since, as the number of of intermediate nodes increases, there is a greater chance to experience more cuts. The equation also shows that N_c is a decreasing function of the link utilization factor, ρ, which means that in a less crowded network the average number of cuts is higher [13].

3.6 DISCUSSION OF RESULTS

Based on the model suggested in [26], a node that includes a communication processor, with fixed processing time, and communication controllers, that manage link traffic, may be modeled as an M/D/1 queueing stage followed by M/M/1 queues. Communication delay, at a node, is the sum of delays due to the communication processor queue and link traffic queues. A balanced network of general topology is considered to derive network end-to-end delay times for the message switching technique and the cut-through switching technique. Analysis of the equations presented in this chapter leads to the following conclusions:

1. Equations 3.3 and 3.5 show that the mean internode distance, N_h, plays a key role in determining a network delay. One way to reduce the impact of high mean internode distance on network delay, particularly at low traffic loads, is to use virtual cut-through switching methodology. Another option is to reduce the mean internode distance itself. The mean internode distance depends on network topology. Several topologies are considered in the next chapter.

2. Network delay times are expected to grow dramatically as traffic densities, represented by λ_{cp} and λ_l, approach message service rates, μ_1 and μ_2, respectively. A faster communication processor offers a high μ_1 and a link of higher bandwidth increases μ_2. To minimize network delay times, it is not enough to only use the best available hardware, if it is affordable. Queue waiting times should also be reduced. This may be achieved through intelligent management of link traffic density and communication processor traffic density. Evaluation of these parameters is discussed in the following chapter.

Because of the large number of symbols and notations used in this chapter, the model parameters are summarized in the following table.

SYMBOL	DEFINITION
N	The total number of nodes in the network.
N_l	The total number of links per node.
N_h	The mean number of hops travelled by a message from source to destination.
N_c	The total number of cut-throughs that can be made.
λ_{fp}	The mean message generation rate by the function processor.
λ_{cp}	The aggregate traffic rate at the communication processor.
λ_l	The aggregate traffic rate on a link.
$\frac{1}{\mu_1}$	The constant service time for message processing.
$\frac{1}{\mu_2}$	The mean message transmission time, header is included in the message length.

Table 1. SUMMARY OF NETWORK MODEL PARAMETERS.

4.0 PERFORMANCE MEASURES FOR STATIC NETWORKS

4.1 INTRODUCTION

In this chapter the strengths and weaknesses of several interconnection network topologies are reviewed. Each is characterized by determining the rates of increase of several key factors as N, the total number of nodes in the network, increases. These factors include mean internode distance and message traffic density. As shown previously, these factors can dramatically affect network delay performance and determine the suitability of various multicomputer designs for different applications.

As discussed in Chapter 2, there are many connection methods, or topologies, for linking networks of computers. Each topology covered in this research consists of active computing nodes connected by passive communication links. All switching occurs at the nodes. This chapter concentrates on hypercube structures, a subset of static interconnection networks. We will not consider reconfigurable multistage switching networks,

such as the Banyan, Omega, and shuffle-exchange. "These networks have most often been suggested for data routing in SIMD (single-instruction-stream multiple-data-stream) machines, although some could be used to interconnect MIMD nodes" [2]. There is a rich literature comparing these reconfigurable interconnection networks. Performance measures are derived for spanning bus hypercube and torus networks under uniform message routing assumption. The torus topology is further analyzed when network traffic follows a sphere of locality message distribution. Performance measures for both topologies are tabulated in Table 2.

4.2 TORUS

As mentioned earlier, a W^D torus topology is a D-dimensional hypercube, of width W, with end-around connections. Each node is connected to its nearest neighbors by separate communication links, as shown in Figure 3. Thus, a torus may be viewed as if each of its dimensions forms a ring of size W. Network size $N = W^D$ may be changed by varying either D or W. Because each of the N nodes is connected to D rings, there are ND communication links and the total connection cost is $2ND$ connections.

To evaluate performance measures, node addresses may be viewed as D digit, base W numbers [3,8]. Each digit represents a ring of W nodes, and a message is routed to its destination by successively sending the message to the correct place on each of the D rings. Under the uniform traffic assumption, messages flow identically in each dimension and from each of the W positions along an axis. The average number of hops in each dimension is [8]

$$N_h \text{ one dimension} = \frac{\sum_{K=0}^{W-1} \min(K, W-K)}{W}$$

$$= \frac{W}{4} \qquad \text{if W is even and}$$

$$= \frac{W^2 - 1}{4W} \qquad \text{if W is odd.}$$

The minimum function in the sum reflects the routing of messages along the shorter of the two potential paths in each ring. Because the dimensions are independent, the mean internode distance in a W^D torus is given by the following equation for W even:

$$N_h = D \times (\frac{W}{4}) \times (\frac{N}{N-1}) \cong \frac{DW}{4} \tag{4.1}$$

The factor $(\frac{N}{N-1})$ occurs because we assume that a node does not route a message to itself. For a symmetric network of identical node and link structures, the link traffic density, λ_l, is obtained for even W as

$$\begin{aligned} \lambda_l &= N_h \times \frac{\text{(total traffic generated)}}{\text{total number of links}} \\ &= \frac{N_h(N\lambda_{fp})}{DN} \\ &= \frac{DW}{4} \times N\frac{\lambda_{fp}}{D} \\ &= \frac{W\lambda_{fp}}{4} \end{aligned} \tag{4.2}$$

A similar expression for λ_l is obtained for the case of odd W and is included in Table 2. The communication processor traffic density, λ_{cp}, is given by

$$
\begin{aligned}
\lambda_{cp} &= \text{traffic per node} + \lambda_{fp} \\
&= \frac{\text{total traffic}}{\text{number of nodes}} + \lambda_{fp} \\
&= \frac{N_h(N\lambda_{fp})}{N} + \lambda_{fp} \\
&= (1 + N_h)\lambda_{fp}
\end{aligned}
\tag{4.3}
$$

The derivation of mean path lengths for non-uniform message routing distributions, though conceptually straight forward, is computationally difficult. To simplify exposition we use the recursive *Reach* function [8] defined below for the torus when its width W is odd:

$$
\begin{aligned}
Reach(L,D,W) &= 1 \quad \text{for } L = 0 \\
&= 2 \quad \text{for } D = 1 \text{ and } L \le |\frac{W}{2}| \\
&= 0 \quad \text{for } D = 1 \text{ and } L > |\frac{W}{2}| \\
&= 2 \sum_{l=1}^{\min[L, |\frac{W}{2}|]} Reach(L - l, D - 1, W) \\
&\quad + Reach(L, D - 1, W) \quad \text{otherwise}
\end{aligned}
\tag{4.4}
$$

where $|\frac{W}{2}|$ denotes the integer portion of $(\frac{W}{2})$, i.e., the largest integer less than or equal to $(\frac{W}{2})$ and $Reach(L,D,W)$ denotes the number of nodes exactly L links away from a source node in a torus of dimension D and width W.

The factor 2 appears in the above equation because of network symmetry. If we assume a source node is at the center of a $(D - 1)$-dimensional hyperplane and has $|\frac{W}{2}|$ $(D - 1)$-dimensional hyperplanes above and below it, then a message can, due to symmetry, go up or down exactly the same number of links away from the source node. In

the case of even W, this inner symmetry is lost and we modify the *Reach* function definition to be:

$$
\begin{aligned}
Reach(L,D,W) &= 1 \quad \text{for } L = 0 \\
&= 2 \quad \text{for } D = 1 \text{ and } L < |\frac{W}{2}| \\
&= 1 \quad \text{for } D = 1 \text{ and } L = |\frac{W}{2}| \\
&= 0 \quad \text{for } D = 1 \text{ and } L > |\frac{W}{2}| \\
&= \sum_{l=1}^{\min[L,\,|\frac{W}{2}|]} Reach(L - l, D - 1, W) \\
&\quad + Reach(L, D - 1, W) \quad \text{otherwise}
\end{aligned}
\tag{4.5}
$$

Once the number of nodes in the sphere of locality is obtained, it is relatively easy to obtain the mean internode distance. The mean number of hops when the message routing distribution follows a sphere of locality distribution with a radius L and probability φ is given by:

$$
\begin{aligned}
N_h &= \varphi E(N_{hops} \le L) + (1 - \varphi)E(L < N_{hops} \le D|\frac{W}{2}|) \\
&= \varphi \sum_{K=1}^{L} KPr(N_{hops} = K) + (1 - \varphi) \sum_{K=L+1}^{D|\frac{W}{2}|} KPr(N_{hops} = K)
\end{aligned}
$$

where $D|\frac{W}{2}|$ is the network diameter of the torus [8]. Using the classical definition of probability [28], the above equation becomes

$$N_h = \varphi \times \frac{\sum_{K=1}^{L} K \times Reach(K,D,W)}{\sum_{K=1}^{L} Reach(K,D,W)} + (1 - \varphi) \times \frac{\sum_{K=L+1}^{D\lfloor \frac{W}{2} \rfloor} K \times Reach(K,D,W)}{\sum_{K=L+1}^{D\lfloor \frac{W}{2} \rfloor} Reach(K,D,W)} \tag{4.6}$$

In the first summation of Equation 4.6, the sum starts from 1. This means that we rule out the possibility that a source node will send messages to itself.

There are two ways to expand a torus network size:

(a) When dimension D is fixed, the width W may grow as $N^{1/D}$. In this case the number of connections per node is constant. The same module can be used regardless of network size. However, both message delay and traffic intensity, in terms of N_h and λ_l , respectively, increase relatively rapidly as $N^{1/D}$.

(b) When W is fixed, D may increase, relatively slowly, as log N. A slow increase in D is desirable because it results in a slow growth in the mean internode distance, N_h , as can be seen from Equation 4.1. However, since expanding the number of dimensions requires two additional link connections per node, it will usually be necessary to rewire the entire interconnection network if this method of expansion is used.

4.3 SPANNING BUS HYPERCUBE

The spanning bus hypercube is a D-dimensional lattice of width W in each dimension. Each node is connected to D buses, each spanning one of the orthogonal dimensions. In each dimension, W nodes share the bus spanning that dimension [2]. An example of the spanning bus hypercube topology is shown in Figure 2. Because each of the W^D nodes is connected to D buses, there are ND connections in a network of size $N = W^D$. Compared to a torus, spanning buses require half as many ports per node.

In each dimension of its path, a message in a W^D spanning bus hypercube needs one hop across the bus to one of the W nodes sharing the bus. Thus for D independent dimensions, the average message distance is given by $N_h = D$. However, the previous relation should be modified by a factor of $\frac{(W-1)}{W}$ to reflect the fact that there are only W nodes along a bus [2]. On the average, in $1/W$ of all cases, the source and destination nodes will be the same and the bus will not be used. This gives the mean internode distance as [2]:

$$N_h = D \times \frac{W-1}{W} \tag{4.7}$$

Since W nodes share each bus, the total number of buses is $\frac{ND}{W}$ and the traffic density for each bus (link) is

$$\lambda_l = \frac{N_h N \lambda_{fp}}{(ND/W)} = D \times \frac{W-1}{W} \times \frac{W}{D} \times \lambda_{fp} = (W-1)\lambda_{fp} \tag{4.8}$$

and λ_{cp} is given by

$$\lambda_{cp} = (N_h + 1)\lambda_{fp} = [1 + \frac{D(W-1)}{W}]\lambda_{fp} \tag{4.9}$$

Like the torus, spanning bus hypercubes built of standard modules have a practical limit to increasing the dimension D because the number of ports per node, D connections for D buses, has to increase. Expanding a spanning bus hypercube network by increasing its width is also restricted. The width W of nodes sharing each bus is limited by the bus bandwidth and by driver/receiver characteristics at each node [2]. The torus topology does not have this limitation because each bus connects two nodes only. In summary, a spanning bus hypercube is harder to extend than a torus. On the other hand, compared to a torus, spanning buses require half as many ports per node. Moreover, the average number of hops along spanning buses is less than that along torus links by a factor of approximately $\frac{W}{4}$ (refer to Equations 4.1 and 4.7).

4.4 SUMMARY

Table 2 summarizes the expressions of N_h, λ_l, and λ_{cp} for different topologies. In this table we define β and γ as the communication processor traffic load factor and link traffic load factor, respectively. β and γ satisfy the following relations

$$\lambda_{cp} = \beta\lambda_{fp}$$
$$\lambda_l = \gamma\lambda_{fp}$$

It is clear, from Equations 4.3 and 4.9, that β is simply equal to $N_h + 1$. The expressions in Table 2 are valid for large N. In the case of a small network the expressions for

N_h and γ should be multiplied by a factor of $(\frac{N}{N-1})$ to reflect the fact that a source node does not send messages to itself.

As can be seen, network topology has mixed effects on network traffic parameters. The mean internode distance is decreased for the spanning bus hypercube topology, compared to the torus, for a price of increased link traffic density. To quantify and examine the effect of these two performance measures, both topologies are implemented in the network analyzer program which is the subject of the next chapter.

Topology	N_h	γ	β
Binary Torus $N = 2^D$	$\frac{D}{2}$	$\frac{1}{2}$	$(\frac{D}{2}+1)$
Torus $N = W^D$ W even W odd	 $\frac{WD}{4}$ $(\frac{W^2-1}{4W})D$	 $\frac{W}{4}$ $(\frac{W^2-1}{4W})$	 $(\frac{WD}{4}+1)$ $[1+\frac{(W^2-1)}{4W}]D$
Spanning Bus Hypercube	$(\frac{W-1}{W})D$	$(W-1)$	$[(\frac{W-1}{W})D+1]$

Table 2. NETWORK PARAMETERS FOR HYPERCUBE STRUCTURES.

5.0 NETWORK PERFORMANCE ANALYSIS TOOL

5.1 INTRODUCTION

In order to assess the performance of different multicomputer network alternatives, the "Network Analyzer" program was developed to serve as a design and analysis tool. The program accepts a network description and computes nodal and link delay times. End-to-end delay times are generated as a function of message generation rate within each node. Network Analyzer is relatively easy to learn and use, and once learned, it is easy to change network parameters to perform "what-if" analysis. The program can also be used to quickly generate numerical results and plots to illustrate the results of the analysis. This chapter provides a brief summary of the program structure and general guidelines for using the program. Chapter 3 provides detailed definitions of the parameters used below.

The program is developed for balanced networks of symmetric topologies. Networks are modelled as an interconnection of identical nodes, where each node includes a communication processor and a number of communication controllers to handle message transfer between nodes. A node is modelled as an M/D/1 stage that represents the communication processor queueing delay and a second stage of several M/M/1 systems that represent the communication controllers and link transmission queues. The model assumes unlimited nodal storage capacity for message queues. Composite arrival traffic at each node is is assumed to have a Poisson distribution. Message lengths, and therefore message transmission times, are assumed to be exponentially distributed. The program is a suitable tool for trade-off studies of different multicomputer interconnection network designs.

5.2 PROGRAM STRUCTURE

A listing of the Network Analyzer program appears in Appendix A. The program is implemented in Turbo Pascal [29]. Turbo Pascal provides a number of compiler directives to control special runtime facilities, such as recursion and user interrupt, as well as plotting facilities. These directives have been activated. The user interrupt directive allows the user to interrupt the program any time during execution by entering a Ctrl-C. The recursive directive was activated and was particularly useful for implementing the recursive *Reach* function defined in Equations 4.4 and 4.5.

The program is divided into three main sections:

(1) parameter selection through the main menu,

(2) computation, and

(3) plotting the computed data.

Such a modular structure is attractive because of ease of program expansion and modification. It also aided in debugging the program during development and testing phases.

5.2.1 Parameter Selection Phase

The main menu offers ten choices and may branch to submenus. Choices are summarized as follows:

1- topology

1.1 binary torus[1]

1.2 W**D torus

1.3 W**D spanning bus hypercube

1.4 others[2]

2- traffic pattern

2.1 uniform

2.2 sphere of locality[3]

3- switching technique

3.1 message switching

3.2 virtual cut-through

4- link bandwidth

5- mean message length

6- mean header length

---

1 The first choice in a submenu is the default value.

2 User provides numerical values for the network parameters, N_h, β, and γ, for "others".

3 Sphere of locality is implemented for the torus topology only.

7- processing time

8- compute

9- plot

10- exit

Using the menu, a user may specify a wide variety of network designs and compute and plot a performance curve for each design. Performance is measured in terms of end-to-end message communication delay. The program uses the network model discussed in Chapter 3. The total message delay for message switching and cut-through switching is given by Equations 3.4 and 3.7, respectively, which are written below for convenience.

$$T_{MS} = (N_h + 1)[\frac{1}{\mu_1} + \frac{\lambda_{cp}}{2\mu_1(\mu_1 - \lambda_{cp})}] + \frac{N_h}{\mu_2 - \lambda_l}$$

$$T_{CT} = T_{MS} - (N_h - 1)(1 - \rho)[\frac{1}{\mu_1} + \frac{\lambda_{cp}}{2\mu_1(\mu_1 - \lambda_{cp})} + \frac{(1 - \alpha)}{\mu_2}]$$

Menu selections 4 through 7 are straight forward and affect message service times as follows:

μ_1 = 1/processing time
μ_2 = transmission rate/mean message length
α = mean header length/mean message length

The last parameter, α, is relevant only if the virtual cut-through technique is used (refer to Equation 3.7). Choices 1 and 2 are tightly related and affect the following parameters:

$\lambda_{cp} = \beta\lambda_{fp}$,
$\lambda_l = \gamma\lambda_{fp}$, and
N_h = average number of hops through the network.

where we define β and γ as the processor traffic load factor and link traffic load factor, respectively. A summary of the expressions for β, γ, and N_h for the implemented topologies and traffic patterns is given in Table 2. Evaluation of these parameters is deferred to the precomputation phase to allow the user to visit the main menu as many times as needed without entering too many parameters each time.

5.2.2 Computation Phase

Before the start of computation, network parameters are evaluated according to the current choices of the main menu items 1 through 7. If either the current topology or traffic patterns are user defined, the user is alerted that he will provide N_h, β, and γ before the start of computation. Currently the program allows up to six curves to be computed and plotted in one session. These curves are labeled sequentially (1,..,6) when computed. If more curves are required, data points of the new curve overwrite data points of the oldest curve. This means that the user will always access the most recently created set of curves. A total of 50 points are computed for each curve. Delay time is computed as a function of λ_{fp} , the packet generation rate by the resident functional processor in each of the network nodes. The parameter λ_{fp} is initially varied from 0 in steps of 100 packets/second. The stepping interval is reduced gradually as λ_{cp} or λ_l grows closer to μ_1 or μ_2, respectively, and the value of $\frac{1}{(\mu - \lambda)}$ tends to grow out of bounds.

Data are stored sequentially in the data file network.dat for possible later processing. In this file, data is listed in two columns using format 2F15.5. Each line represents a point (x,y) on a curve with 50 points per curve. Curves are numerically indexed before each computation. The user should note the indices as they are used to select which curves to plot.

5.2.3 Plot Phase

Before curves are plotted, the user is prompted to select whether to plot all curves or selectively choose the curves one by one. The user is also able to select scales for λ_{fp}, packets/second, and network delay, in milliseconds, by entering maximum values for each.

5.3 USER GUIDE

5.3.1 What The User Needs

The Network Analyzer requires the following:

1. An IBM-PC or compatible with at least one floppy disk drive, 256K of memory, and DOS 2.0 or higher.

2. One of the many standard display adapters for the IBM-PC.

5.3.2 What The User Must Know

The user must know enough about DOS to boot the system and enter commands, and about network model parameters, particularly N_h , β, and γ defined earlier.

5.3.3 Using The Program

The following steps can be used to start the program:

1. Insert diskette in drive A.

2. Boot DOS.

3. Type Graphics.

4. Type Net_Analyzer.

5. Now the program starts and the main menu is displayed.

6.0 APPLICATION EXAMPLES

6.1 INTRODUCTION

In this chapter, the performance models are applied to several multicomputer designs to investigate the impact of different network parameters on network delay and demonstrate the effectiveness of the Network Analyzer program as a design and analysis tool. The node model discussed in this research assumes a node structure that includes a communication processor, with fixed message processing time, and several communication controllers that manage message transfer across communication links. Network delay is developed for a network of general topology when either message switching or virtual cut-through switching is used. The torus and the spanning bus hypercube topologies are implemented in the program. Both topologies may be studied assuming a uniform traffic load across the entire network. The torus can also be analyzed for the case of sphere of locality traffic pattern. The program also allows the study of the effects of other network parameters, for example message length, message processing time, and link bandwidth, on network delay.

6.2 EXAMPLE NETWORK

For simplicity, a multicomputer network with a binary torus topology is considered first. The effect of topology on network delay is discussed later. The example network considered is the network described by the default values of various network parameters that are implemented in the Network Analyzer program. The network is assumed to have a binary torus topology of dimension $D = 10$, i.e. network size is $N = 1024$ nodes, and the mean internode distance is $N_h = 5$ links (refer to Table 1), assuming a uniform traffic distribution. It is assumed that messages have an average length of 512 bytes. Message length includes the header and a header length of 26 bytes is assumed. Message processing time by the communication processor is assumed to be 0.1 milliseconds while link bandwidth, BW, is considered to be 10 megabits/second. Note that such a bandwidth results in an average message transmission time, $\frac{1}{\mu_2}$, of about 410 microseconds obtained as follows

$$\begin{aligned} \frac{1}{\mu_2} &= \text{average message transmission time} \\ &= \frac{\text{average message length}}{\text{link bandwidth}} \\ &= \frac{512 \times 8}{10^6} \\ &= 410\ \mu\text{sec.} \end{aligned} \tag{6.1}$$

where μ_2 is the mean message service rate of a link queue, as defined in Chapter 3.

The above network description is practically feasible with current technology. Hypercubes have already been built with 1024 nodes [10]. A mean message length of 512 bytes is a bit generous but can always be changed to reflect the desired granularity. A

header length of 26 bytes results in a useful message utilization of about 95% indicating a reasonable message overhead. Links with 10 Mbits/sec are already implemented in the iPSC [30]. Message processing time depends on the complexity of the message routing algorithm and the communication processor cycle frequency or speed. If M is the number of processor cycles or microinstructions executed to process one message and P is the processor cycle frequency, the message processing time is given by $\frac{M}{P}$ [1]. M is typically a few hundred cycles or less [1].

6.3 NETWORK DELAY ANALYSIS

Figure 9 shows network delay times for the above network when a uniform message routing distribution is assumed and message switching is used. Delay times are derived from Equation 3.3 and reflect different link bandwidths and message processing times. Communication delays are shown for the network described earlier and when either the message processing time or the link bandwidth is doubled.

The plotted results show the magnitude of the communication processing delays relative to transmission delays. One obvious conclusion is that communication processing times become dominant at high message generation rates. In fact, traffic rate thresholds, the traffic rates beyond which delay becomes very large, are determined solely by queueing delays at the communication processor. The figure shows that doubling the message processing time results in lowering the traffic threshold by a factor of two, from 1600 packets/sec to about 800 packets/sec. Moreover, processing delays are comparable to transmission delays at low traffic rates even when the average communication processor time per message is as low as 0.1 millisecond, in spite of the generous

assumption about message length. Doubling the link bandwidth, while keeping the message processing time at 0.1 millisecond, reduces network delay by a factor of $\frac{1}{3}$ only over a packet generation rate, λ_{fp} , of 0 to 1200 packets/sec.

Note that using an average message length of 256 bytes, instead of 512 bytes, not shown in Figure 9, results in network delays that are identical to delays obtained when link bandwidth is doubled. This is expected because either change has the same effect on average message transmission time, $\frac{1}{\mu_2}$, as obtained from Equation 6.1.

Figure 10 shows delay times for the example network when traffic follows a uniform distribution. Results are plotted when either message switching or virtual cut-through switching is used. The results are computed from Equations 3.3 and 3.7. It is observed that the virtual cut-through delay is always less than the message switching delay, particularly at low traffic loads. As traffic grows the number of cut-throughs is decreased and the cut-through technique loses its advantage. Minimal increases in network delay, at low traffic rates, are encountered when the header length is doubled for the same message length. Minimal delay increases indicate that if the header length has to be increased to accommodate virtual cut-through switching, the impact of the header length on network delay is negligible.

The effects of traffic pattern changes on average message delay are displayed in Figures 11 and 12. In Figure 11, the probability of sending a message from a source node to nodes within the sphere, φ , is fixed and the effect of increasing the sphere radius is studied. The number of nodes within the sphere is obtained using Equation 4.4 and is used to obtain the average number of hops, N_h , using Equation 4.6. Average message delay is highly dependent on the value of N_h, as evidenced by Equations 3.3 and 3.7.

Figure 11 shows that for a given probability, higher than 0.5, average message delay increases as the radius of the sphere, L, is increased. With $\varphi \geq 0.5$, a higher L means that an average message is more likely to travel a longer distance to reach its destination.

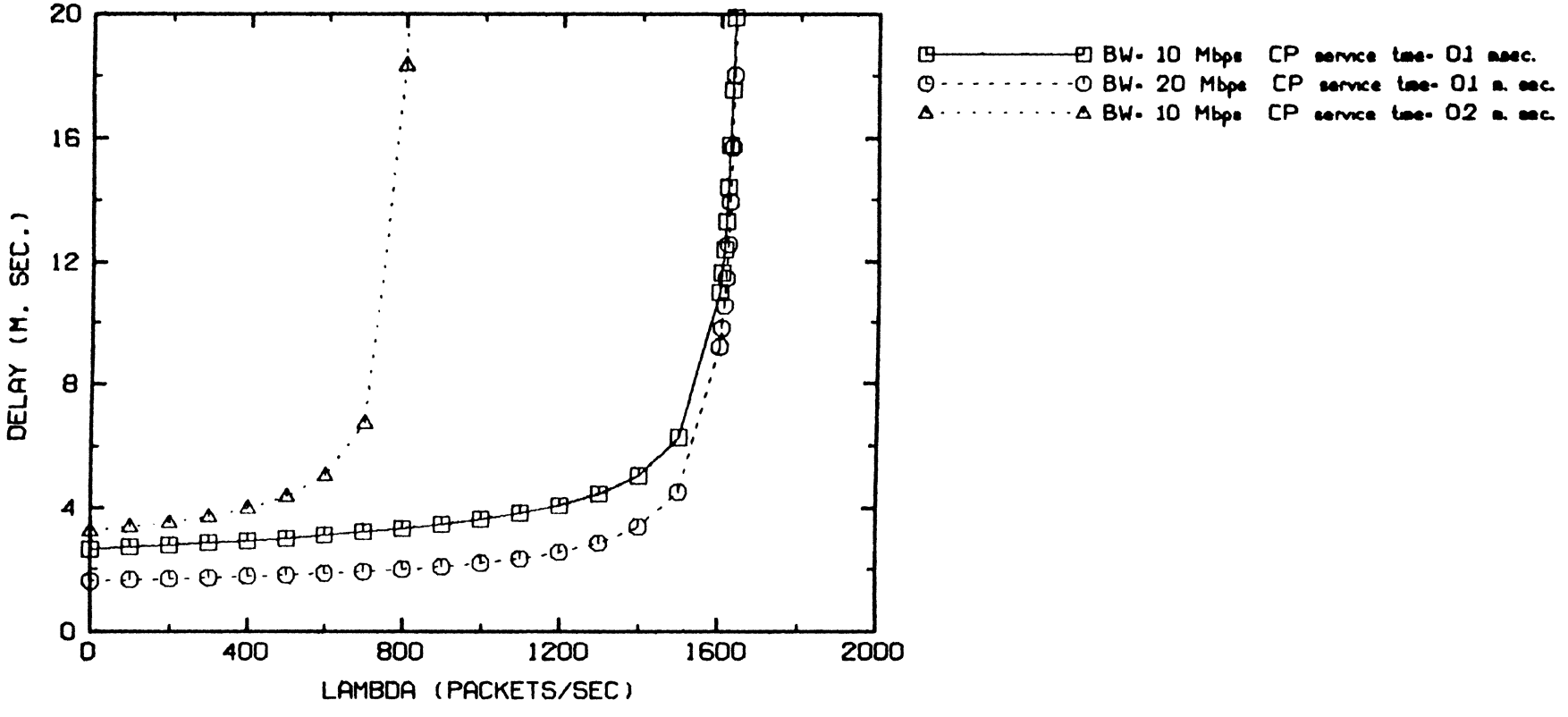


FIGURE 9. MESSAGE DELAY FOR A MESSAGE SWITCHED BINARY TORUS NETWORK.

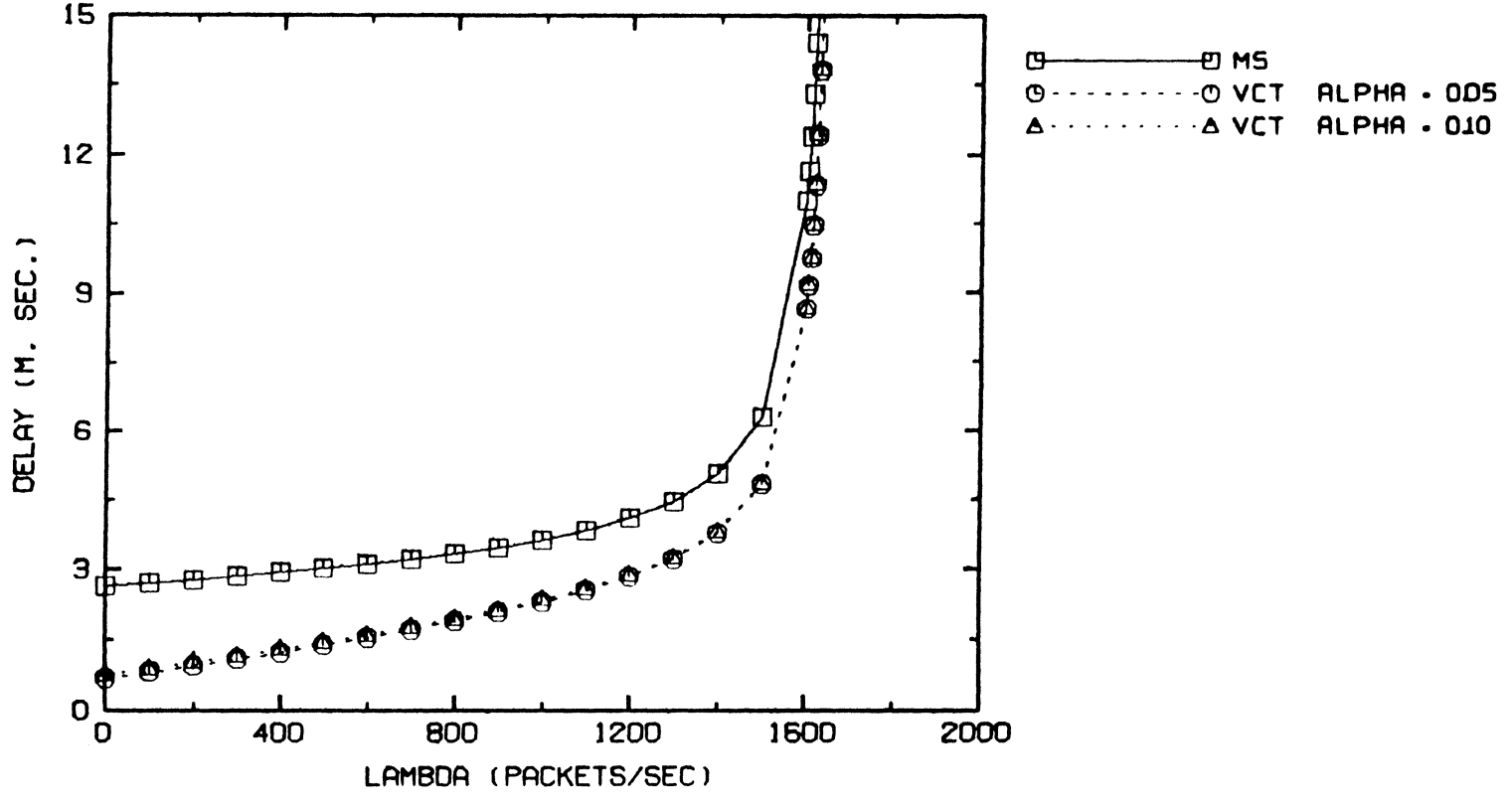


FIGURE 10. NETWORK DELAY FOR DIFFERENT SWITCHING TECHNIQUES.

Figure 11 also shows that for some values of L and φ, in the figure $L = 2$ or 4 when $\varphi = 0.8$, small mean internode distances are obtained and message delay is less for a spherical distribution than it is for a uniform distribution. Locality is obviously useful and should be considered when programs are compiled and tasks are assigned in a multicomputer system.

Similar conclusions may be drawn from Figure 12. Here the sphere radius is fixed at $L = 5$ while probability is changed. Message delay increases as the probability of sending messages further away from a source node is increased. Note that when the probability $\varphi = 1$ and the sphere radius is equal to the network diameter, network delay is equal to that of the uniform distribution. This is because a probability of 1 means that all messages generated will be sent to destinations within the radius.

Finally, the impact of topology on network delays is discussed below. Figure 13 shows network delays for four possible torus topologies that may be used to implement a network of size $N = 4096$ nodes. Uniform message distribution is assumed. The results show that the best performance is achieved when the network width W is minimum and that performance deteriorates as W is increased. This behavior may be explained by the nature of the torus topology. A torus of size $N = W^D$ may be viewed as if each node is connected to a ring of size W and each node is connected to D rings. Increasing W, and thereby decreasing D, for a fixed network size, is equivalent to increasing the size of each ring and reducing the number of rings attached to each node. Increasing a ring size implies an increase in the number of links a message has to travel between two opposite nodes in the ring and leads to an increase in N_h. Reducing the number of rings per node reduces the connectivity between network nodes, again increasing N_h. This explains the binary torus's performance advantage. Note that network delay increases slightly when W is changed from 2 to 4. This is because the mean internode distance is the same for both topologies, but traffic density for $W = 4$ is twice that of a binary torus (refer to

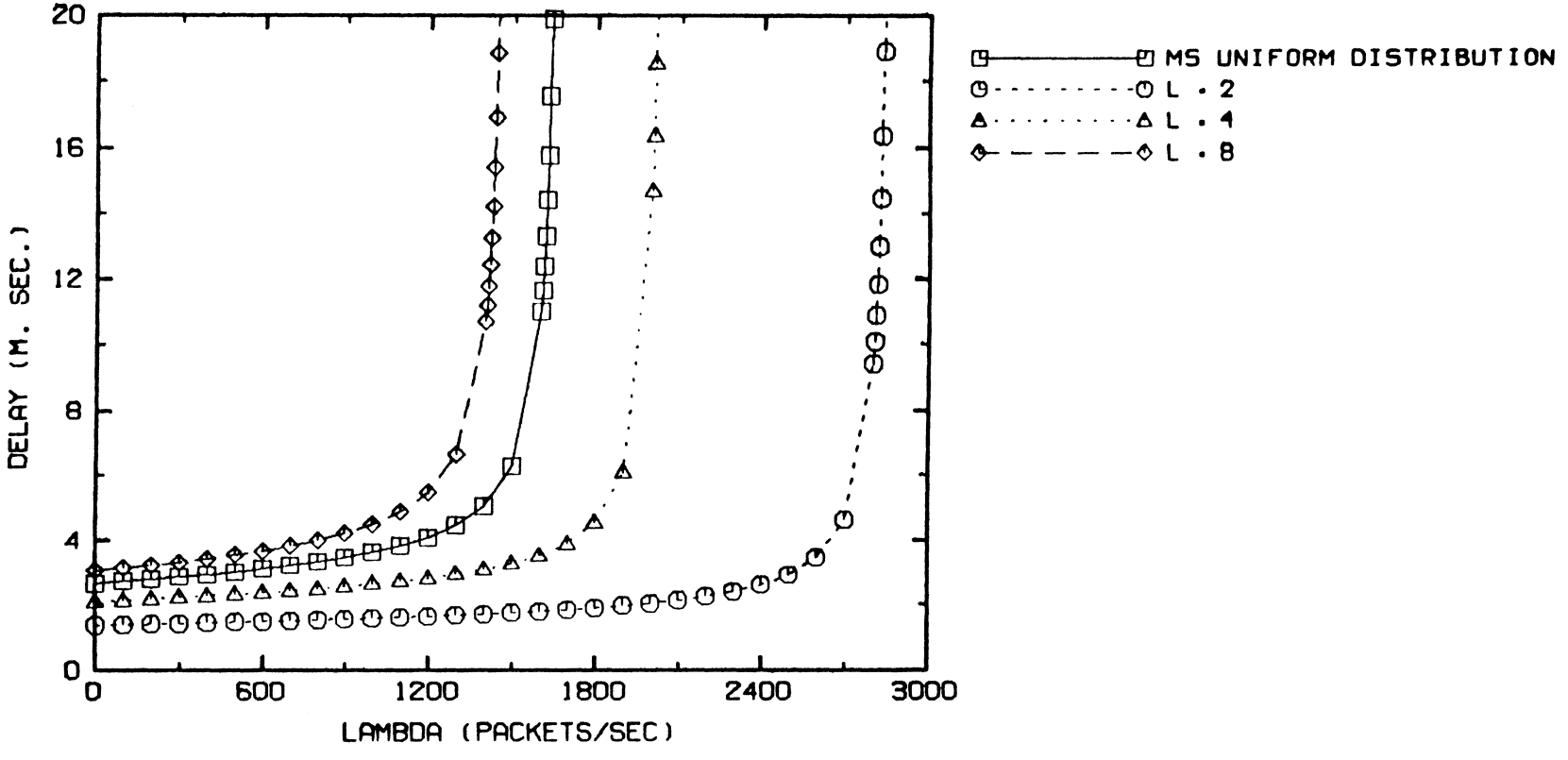


FIGURE 11. EFFECT OF SPHERE RADIUS ON MESSAGE DELAY PROBABILITY = 081

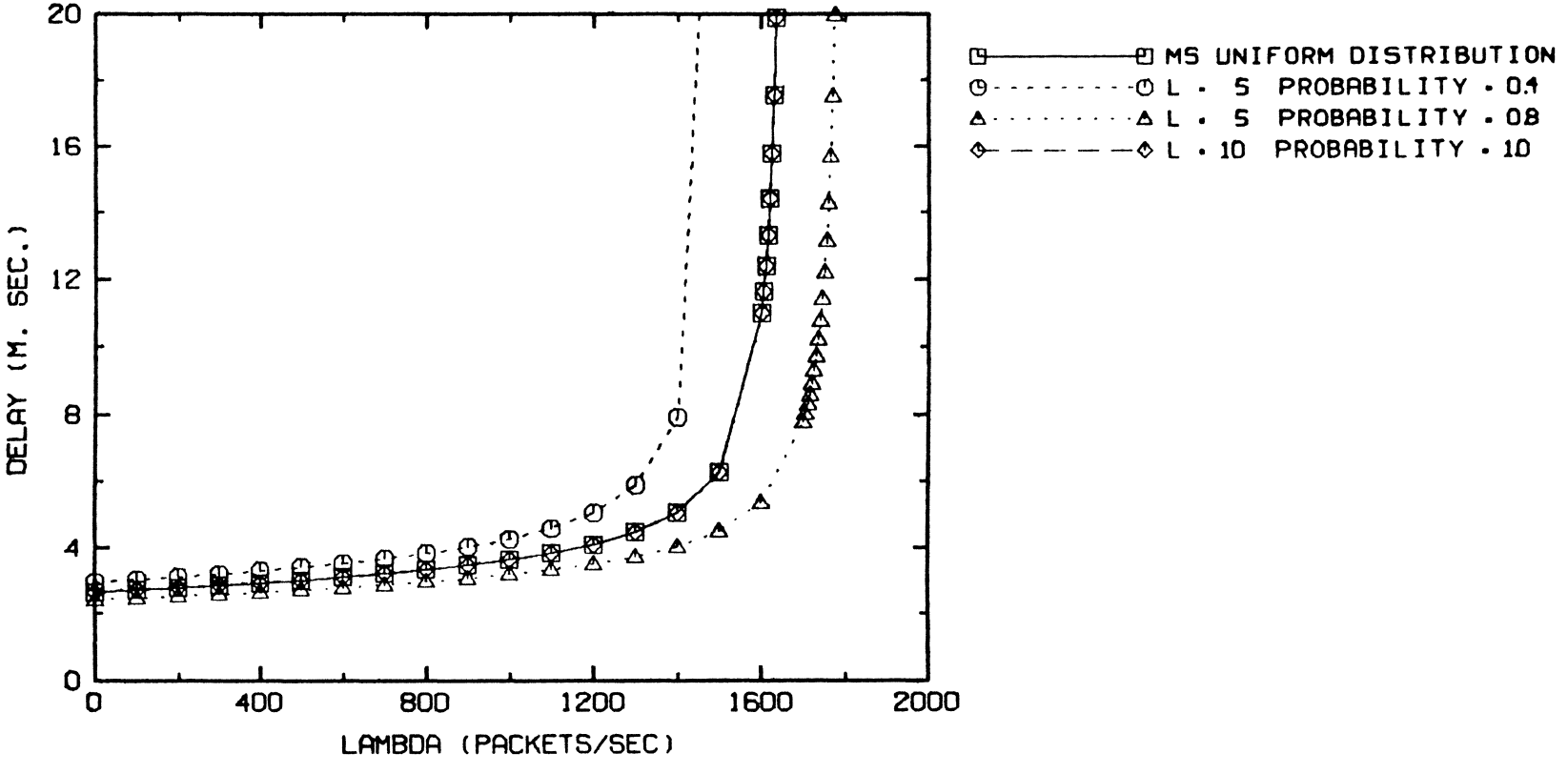


FIGURE 12. EFFECT OF TRAFFIC PATTERN ON MESSAGE DELAY.

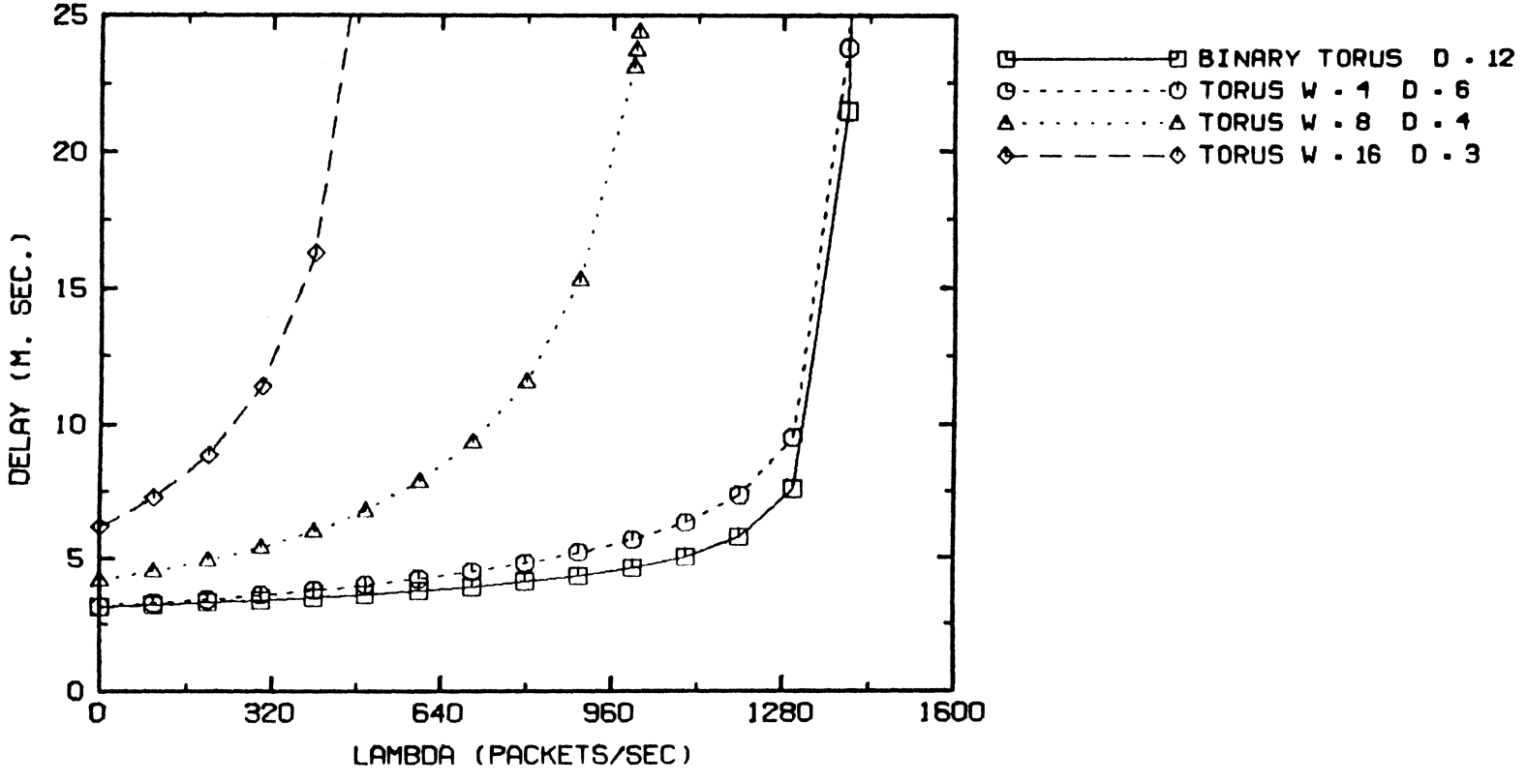


FIGURE 13. NETOWRK DELAYS FOR DIFFERENT TORUS TOPOLOGIES.

Table 2). This also shows how little message transmission time affects total message delay in comparison to message processing time.

In Figure 14, the performance of torus and spanning bus hypercube topologies are compared. Message switching and uniform traffic distribution are assumed. The plotted results show that a spanning bus network offers lower network delays at low traffic loads, but traffic thresholds are reached much earlier than for a torus network of the same width and dimension. Spanning bus topologies have lower delays at low traffic, compared to torus topologies, because they have lower mean internode distance. On the other hand, because spanning bus hypercubes have fewer links, average link traffic density increases rapidly as the traffic load increases.

6.4 NETWORK DESIGN EXAMPLE

In the previous sections, the effects of individual network parameters on network delay performance were investigated. In the following we consider a hypothetical network design problem of a more general nature to demonstrate the effectiveness of the Network Analyzer program as a design tool.

Assume a multicomputer system with the following constraints:

1. Network size is $N = 1024$ nodes. Due to expected growth in system capacity, network size may be increased up to 4096 nodes.

2. Average message length is 512 bytes.

3. Traffic follows a uniform distribution.

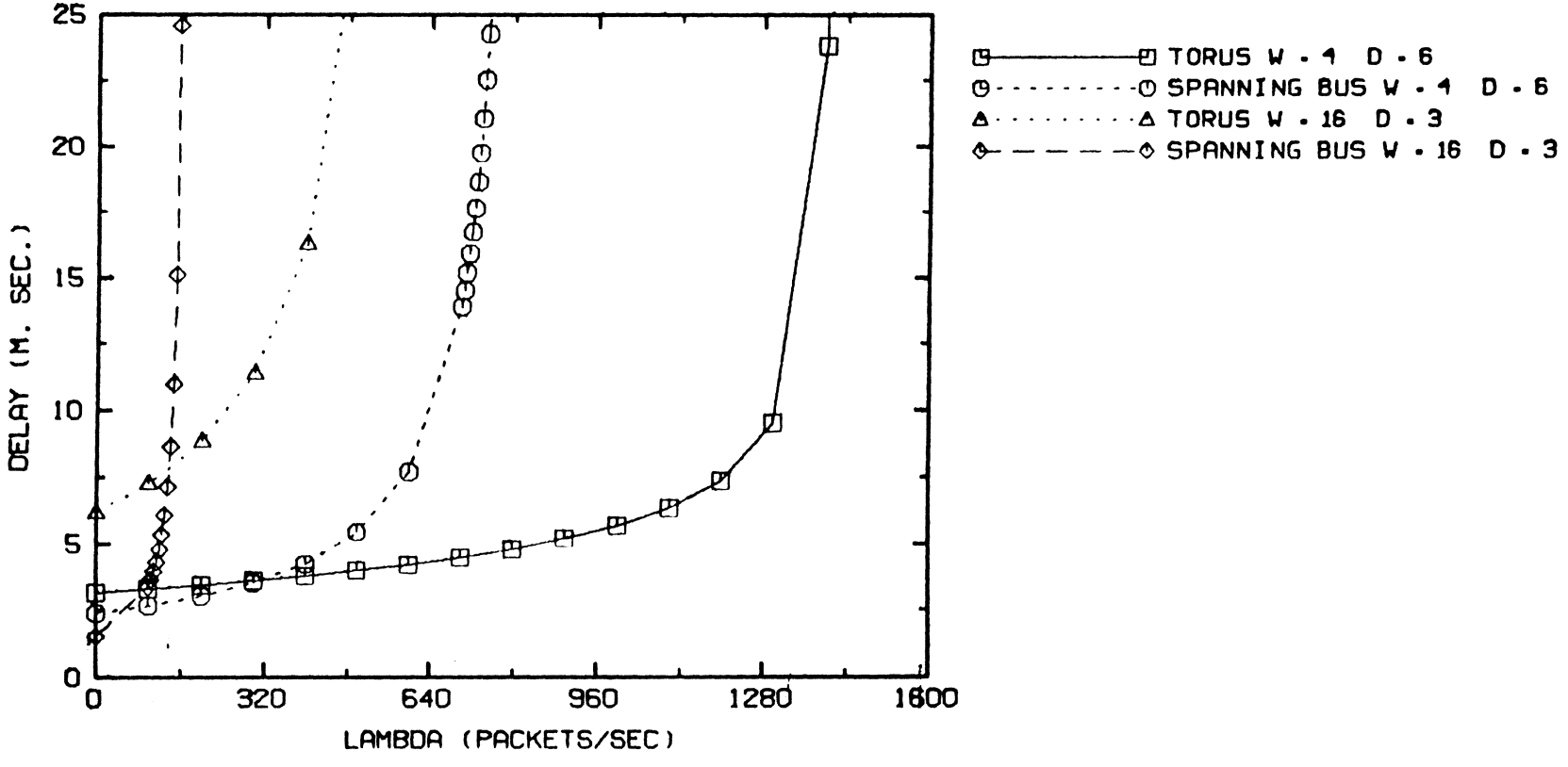


FIGURE 14. PERFORMANCE COMPARISON OF TORUS AND SPANNING BUS HYPERCUBES.

4. Message routing algorithm is simple enough to assume a message processing time of 0.1 milliseconds.

5. Maximum expected message generation rate will not exceed 1000 packets/sec.

It is required to select the most economical topology arrangement that can be used to implement the system without compromising network delay performance.

To simplify matters, it is assumed that network cost is proportional to the following cost function C:

$$C = (BW) \times (N_{cn}) \times (N_l)$$

where BW is the link bandwidth, N_{cn} is the number of connections per node, and N_l is the total number of links in the network.

Using the cost function defined above, one can estimate system cost for torus and spanning bus hypercube topologies as (refer to Chapter 4):

$$C_{\text{torus}} = (BW) \times (2D) \times (ND)$$

$$C_{\text{spanning bus}} = (BW) \times (D) \times (\frac{ND}{W})$$

$$C_r = \frac{C_{\text{torus}}}{C_{\text{spanning bus}}}$$

where C_r is defined as the relative cost of torus to spanning bus topologies.

The above relations show that for a given bandwidth, a spanning bus hypercube offers a cost advantage of $2W$ over a torus topology of the same width and dimension. To make use of that fact, one may consider a bandwidth of 40 megabits/second for the spanning bus topology versus 10 megabits/second for the torus's bus. With the different

bandwidths in consideration, C_r is given by $C_r = \frac{W}{2}$. For ease of physical implementation, a network should be built of regular and identical modules. Because the initial size of the network is $N = 2^{10} = 1024$ and the maximum size is $N = 2^{12} = 4096$, network width may be either $W = 2$ or $W = 4$.

Figure 15 shows message delays for torus and spanning bus hypercube networks when $W = 2$, note that $C_r = 1$ in that case. BW equals 10 Mbit/sec for the torus network and 40 Mbit/sec for the spanning bus network. Communication delays are shown for the minimum and the maximum expected network sizes. It is obvious that the performance of the spanning bus topology is superior. Network delays are consistently lower than those offered by the torus topology for λ_{fp} less than 1000 packets/sec.

Figure 16 shows message delays under the same assumptions of the previous figure except that network width is changed to $W = 4$. The superiority of spanning bus topology is still sustained. This is partly because, as we found in the previous section, the performance of the torus topology deteriorates as W is increased. The interesting observation is that the spanning bus's performance improves when W is changed from 2 to 4. Average message delay is decreased and the permissible packet generation rate is increased from 1200 packets/sec to about 1600 packets/sec. Again, this behavior may be explained by the dependence of the mean internode distance and the link traffic density on W and D as shown in Table 2. Note that even though $C_r = 2$ for $W = 4$, the absolute cost of the spanning bus network is halved when W is changed from 2 to 4.

The results discussed above conclude that a spanning bus topology with $W = 4$ is the best choice to implement the required network in terms of network delay performance and the defined cost function. Another consideration that favors the spanning bus topology is the relative ease of gradual network expansion. Expanding a spanning bus hypercube requires only addition of extra modules along each dimension while expansion of a torus network requires breaking existing connections and inserting new nodes.

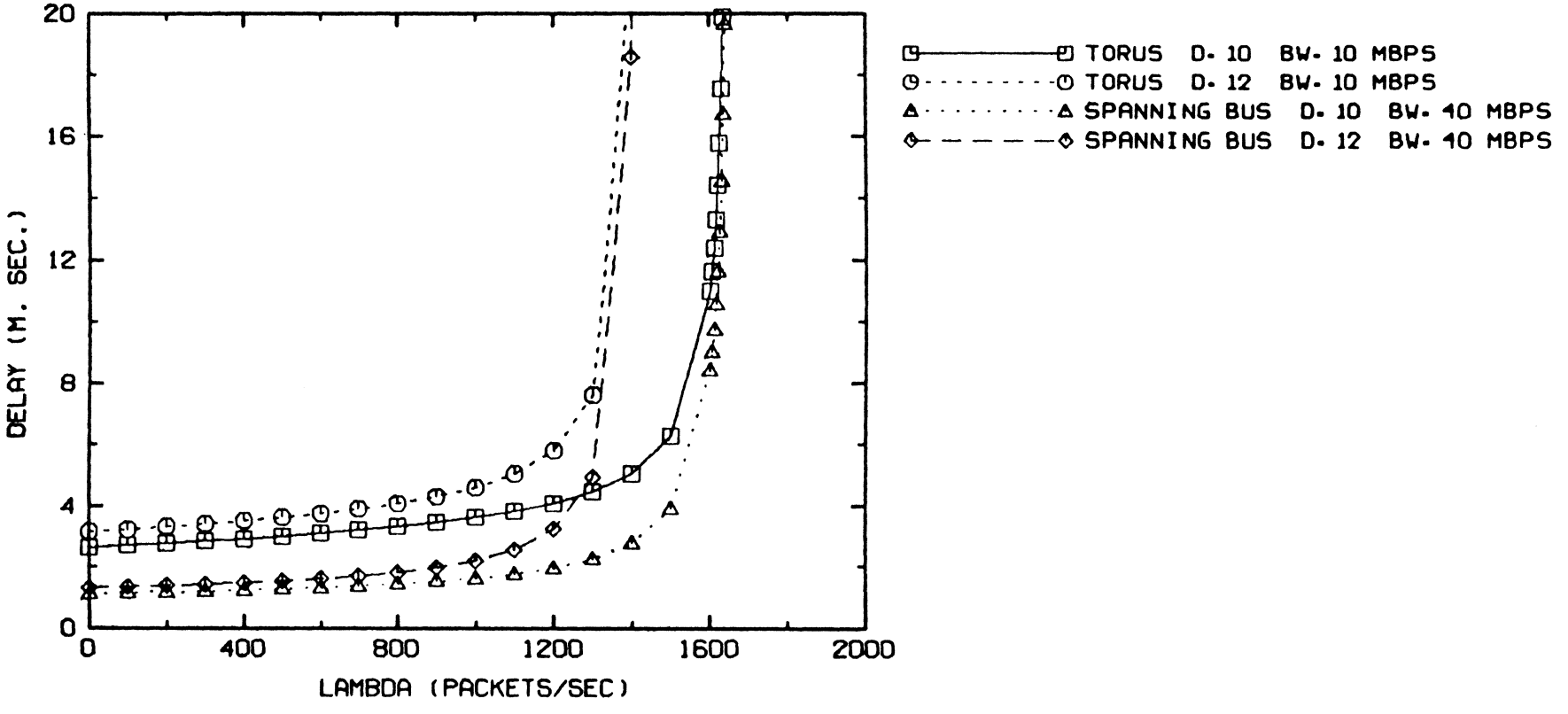


FIGURE 15. PERFORMANCE COMPARISON OF TORUS AND SPANNING BUS TOPOLOGIES W= 21

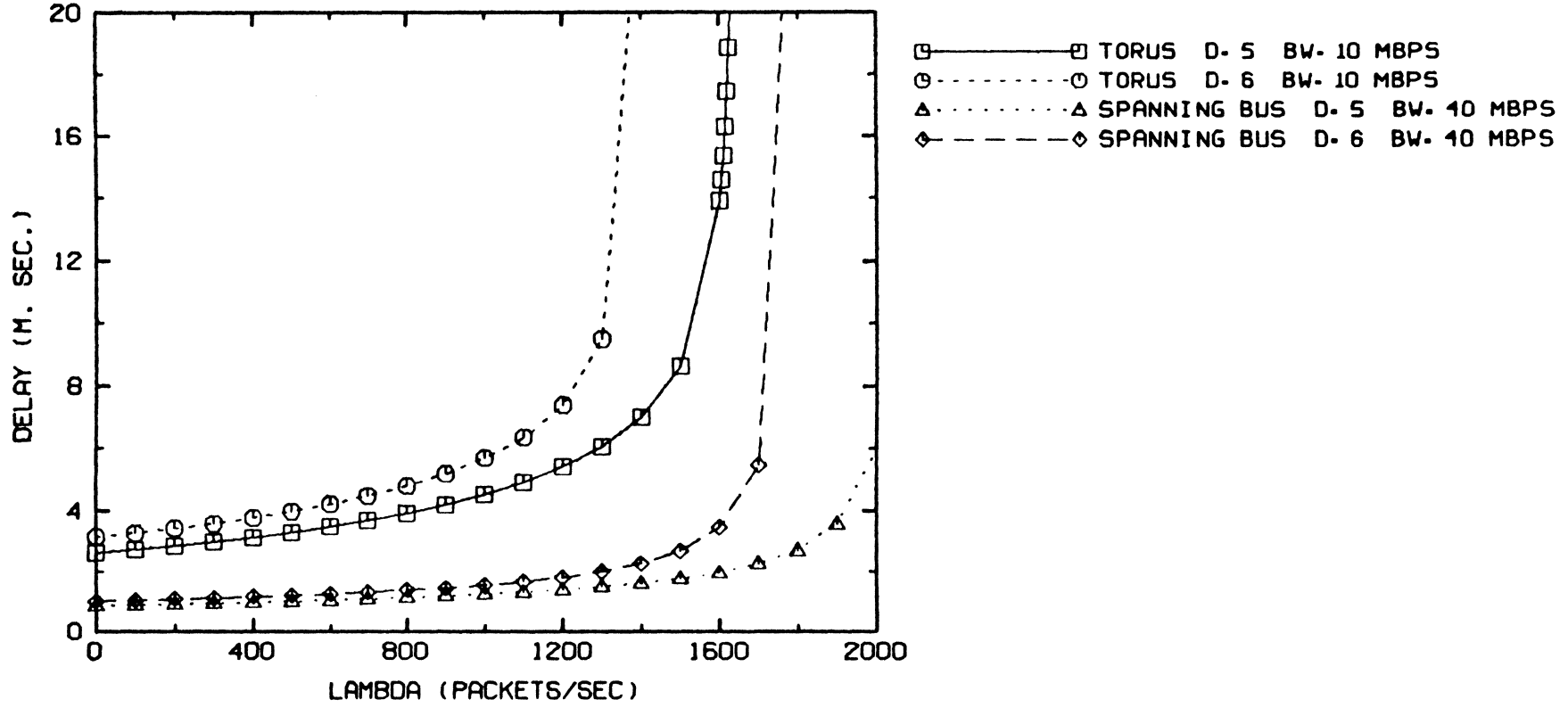


FIGURE 16. PERFORMANCE COMPARISON OF TORUS AND SPANNING BUS TOPOLOGIES W- 41

7.0 CONCLUSIONS

7.1 PURPOSE OF RESEARCH

The goal of this research was to evaluate the performance of multicomputer networks that use point-to-point and spanning bus interconnection topologies, and to develop a design aid that can help explore design alternatives. Performance was measured as the average end-to-end delay across the network. This thesis focused on the impact of network topology, switching technique, and traffic patterns on system cost and performance.

Existing performance models were generalized and modified to model both message switched and cut-through switched networks with different message routing distributions. Virtual cut-through was introduced in [13] and modelled for computer networks where message transmission times represent the bulk of communication delays. Queueing delays at network nodes were represented by M/M/m queues in [13]. The model developed in [26] assumes a message switched network with uniform traffic distribution and considers a node structure that is more representative of nodes in large scale multicom-

puter networks. In such networks, traffic loads tend to be high, message lengths are short, and message routing algorithms are complex enough to justify allocating a special communication processor to manage message transfer functions. The model in [26] also assumes bidirectional links between communicating nodes. In this research, the model in [26] was generalized to allow the use of unidirectional links, as suggested in [27], and extended to represent message delays in balanced networks that may implement either message or cut-through switching with different traffic patterns.

The models were used to develop a network performance analysis program. The program accepts a network description and provides graphs and listings of analysis results. The program consists of three main phases. The first phase allows a user to describe a network design by choosing its parameters through different menus. In the second phase, network delays are computed using the models that were developed. The third phase provides graphical results of the analysis and allows the user to redefine network parameters. The program proved to be a useful design tool and was used to analyze the impact of different network parameters and design decisions on network delays.

The binary torus topology proved to offer minimum network delay over a wide range of traffic loads and patterns in a multicomputer system. Spanning bus hypercube topologies have low average delays at low traffic rates but link traffic saturates quickly afterwards creating a communication bottleneck and increasing communication delays substantially.

Methods to reduce communication delays in torus networks were investigated. One way is to use the cut-through switching technique. Cut-through switching reduces network delay at low traffic rates and approaches message switching performance only at high traffic loads when links begin to saturate. Another method is to assign tasks in multicomputer networks such that traffic patterns have locality. It is shown that locality

in traffic patterns can reduce network communication delays, compared to uniform distribution, at all traffic loads.

7.2 FUTURE RESEARCH

As a basis for future research, this thesis provides an introduction to static large scale interconnection networks. It exposes some fundamental concepts and methods in modelling and analysis of multicomputer network performance and provides a network design program. Additional topologies and design alternatives need to be considered and added to the program to make "Network Analyzer" a more powerful tool for the design engineer.

Topologies like the dual-bus hypercube, the cube-connected cycle, and generalized hypercubes may be considered. Each of these topologies can be analyzed to determine its key parameters such as mean internode distance, link traffic density, communication processor traffic density, system connection cost, and network expansion potential.

Currently, the program allows the study of torus networks with either a uniform or a sphere of locality distributions. In light of the method used in this thesis to determine the number of nodes in a sphere of locality and the mean internode distance, it is possible to apply the sphere of locality distribution to other topologies. Another distribution that is worth investigating is the sphere of locality distribution with decreasing probability [8]. In this distribution, the probability of sending a message decreases as the distance of the destination node from the source node increases.

Finally, it will be useful to add other performance measures to the program in addition to the current one that measures network delay versus packet generation rate.

Different network cost and/or performance functions may be studied versus network size, message length, or message routing distributions.

REFERENCES

1. S.F. Midkiff and C.R. Carroll, "A Communication Processor for Point-to-Point Multiprocessor Networks," *Proceedings Sixth International Phoenix Conference on Computers and Communications,"* 1987, pp. 14-17.

2. L.D. Wittie, "Communication Structures for Large Networks of Microcomputers," *IEEE Transactions on Computers,* vol. C-30, April 1981, pp. 264-273.

3. L.N. Bhuyan, and D.P. Agrawal, "Generalized Hybercube and Hyberbus Structures for a Computer Network," *IEEE Transactions on Computers,* vol. C-33, April 1984, pp. 323-333.

4. C. Wu and T. Feng, *Tutorial: Interconnection Networks for Parallel and Distributed Processing,* IEEE Computer Society Press, Silver Spring, MD, 1984.

5. T. Feng, " A Survey of Interconnection Networks," *IEEE Computer,* vol. 14, Dec. 1981, pp. 12-27.

6. K. Hwang and F.A. Briggs, *Computer Architecture and Parallel Processing,* McGraw-Hill, New York, 1984, Chapter 5.

7. E.M. Aupperele, "MERIT Computer Networks: Hardware Considerations," in *Computer Networks,* R. Rustin, ed., Prentice-Hall, Englewood Cliffs, N.J., 1972, pp. 49-63.

8. D.A. Reed and D.C. Grunwald, "The Performance of Multicomputer Interconnection Networks," *IEEE Computer,* vol. 20, June 1987, pp. 63-73.

9. B.W. Arden and H. Lee, "Analysis of Chordal Ring Network," *IEEE Transactions on Computers,* vol. C-30, April 1981, pp. 291-295.

10. P. Wiley, "A Parallel Architecture Comes of Age at Last," *IEEE Spectrum,* vol. 29, June 1987, pp. 46-50.

11. F.P. Preparata and J. Vuillemin, "The Cube-Connected Cycles: A Versatile Network for Parallel Computation," *Communications ACM,* vol. 24, May 1981, pp. 300-309.

12. W. Stallings, *"Data and Computer Communications,"* Macmillan, New York, 1984.

13. P. Kermani and L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks,* vol. 3, Sept. 1979, pp. 267-286.

14. F.S. Wong and M.R. Ito, "A Loop-Structured Switching Network," *IEEE Transactions on Computers*, vol. C-33, May 1984, pp. 450-455.

15. S. Nakamura and G.M. Masson, "Lower bounds on Crosspoints in Concentrators," *IEEE Transactions on Computers*, vol. C-31, Dec. 1982, pp. 1173-1178.

16. H.J. Siegel, "Analysis Techniques for SIMD Machine Interconnection and the Effects of Processor Address Masks," *IEEE Transactions on Computers*, vol. C-26, Feb. 1977, pp. 153-161.

17. H.J. Siegel, "A Model of SIMD Machines and a Comparison of Various Interconnection Networks," *IEEE Transactions on Computers*, vol. C-28, Dec. 1979, pp. 907-917.

18. G.D. Adams and H.J. Siegel, "On the Number of Permutations Performable by the Augmented Data Manipulator Network," *IEEE Transactions on Computers*, vol. C-31, April 1982, pp. 270-277.

19. L.N. Bhauyan and D.P. Agrawal, "Design and Performance of a General Class of Interconnection Networks," *IEEE Transactions on Computers*, vol. C-32, Dec. 1983, pp. 1091-1098.

20. M.Lee and C. Wu, "Performance Analysis of Circuit Switching Baseline Networks," *Proceedings International Symposium on Computer Architecture*, 1984, pp. 82-90.

21. Y-C. Jenq, "Performance Analysis of a Packet Switch Based on a Single-Buffered Banyan Network," *IEEE Journal on Selected Areas in Communication*, vol SAC-1, Dec. 1983, pp. 1014-1021.

22. D.H. Lawrie and D.A. Padua, "Analysis of Message Switching with Shuffle-Exchanges in Multiprocessors," *Proceedings Workshop on Interconnection Networks for Parallel Distributed Processing*, 1980, pp. 116-123.

23. C.-Y. Chin and K. Hwang, "Connection Principles for Multipath Packet Switching Networks," *Proceedings International Symposium on Computer Architecture*, 1984, pp. 99-109.

24. M. Arango, H. Badr, and D. Glerenter, "Staged Circuit Switching," *IEEE Transactions on Computers*, vol. C-34, Feb. 1985, pp. 174-180.

25. L. Kleinrock, *"Queueing Systems, Vol. II: Computer Applications,"* John Wiley & Sons, New York, 1976.

26. D.A. Protopapas and J.N. Denenberg, "A New Model for Performance Analysis of Large Scale Multicomputer Networks," *Proceedings Sixth International Phoenix Conference on Computers and Communications*, 1987, pp. 451-456.

27. S.F. Midkiff, "Link Allocation in Point-to-Point Multicomputer Networks," to appear, *Proceedings Seventh International Phoenix Conference on Computers and Communications*, 1988.

28. A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, McGraw-Hill, New York, 1984, Chapter 1.

29. Borland International, Inc., *Turbo Pascal Reference Manual,* Scotts Valley, CA, 1986.

30. Intel Corporation, "iPSC Data Sheet," Beaverton, OR, Order No. 280101-001.

Appendix A. PROGRAM LISTING

```
(*****************************************************************************)
(*                                                                           *)
(*                    Program Network Analyzer                               *)
(*                                                                           *)
(*  This program is intended to serve as a Network Analyzer. Network         *)
(*  is modeled as identical nodes, each node consists of two stages:         *)
(*  M/D/1 that represents the Communication Processor (CP) and               *)
(*  M/M/1's that represent the Communication Controllers (CC).               *)
(*  The model assumes the following ASSUMPTIONS:                             *)
(*     1- Composite arrival traffic at each node is Poisson distributed.     *)
(*     2- Message lengths, and therefore message transmission times,         *)
(*        are exponentially distributed.                                     *)
(*     3- Infinite nodal capacity for message queues.                        *)
(*     4- Symmetric network topologies.                                      *)
(*     5- Identical node structure, all communication processors and         *)
(*        links are the same.                                                *)
(*     6- Uniform message generation rate.                                   *)
(*     7- Given assumptions 4,5, and 6 above, traffic patterns cause         *)
(*        uniform load conditions, i.e., the patterns are such that          *)
(*        message rate over each link is the same.                           *)
(*                                                                           *)
(*****************************************************************************)


{$U+}{directive U+ allows program interrupt with Ctrl-C}
{$A-}{directive A- allows recursion}


Program Network_Analyzer;

Type
  Topology = string[28];
  Traffic = string[28];
  Switching = string[28];

Const
  Blank  = ' ';
  Curve_Max = 6;{maximum number of curves in one session}
  Num_Point = 50;
  Step = 100.;
  Delay_Max = 45.;

Var
  Topology_Index: integer;
  Topology_Type : Array [1..4] of Topology;
  Traffic_Index: integer;
  Traffic_Pattern: Array[1..3] of Traffic;
  Switching_Index: integer;
  Switching_technique: Array[1..2] of Switching;
  X : Array[1..Num_Point] of Real;
  Y : Array[1..Num_Point] of Real;
```

```
  Xglobal: Array[1..Curve_Max,1..Num_Point] of Real;
  Yglobal: Array[1..Curve_Max,1..Num_Point] of Real;
  Xtemp: Array[1..Curve_Max,1..Num_Point] of Real;
  Ytemp: Array[1..Curve_Max,1..Num_Point] of Real;
  Legend: Array[1..Curve_Max] of Integer;
  Curve_Index,Temp_Curve_Index: Integer;
  Curve_Max_Reached : String[3];

              {plot parameters}
  Xaxl,Yaxl: integer;
  xtic,ytic,l:Integer;
  xdiv,ydiv:Integer;
  Title: String [50] ;

  X_Max,Y_Max: Integer;
  Fil:       Text;{data file variable}

Var
  User_Choice, Choice: Integer;
  L, D, Num_Points: Integer;
  N_hops, Prob : Real;
  Network_Size: Real;
  Network_Width, Network_Dimension, Network_Diameter: Integer;
  Message_Length, Header_Length: Integer;
  Band_Width, Processing_Time: Real;
  Beta, Gamma: Real;
```

```
(*****************************************************************************)
(*                                                                           *)
(*                    Reach Function                                         *)
(*                                                                           *)
(*  This recursive function is used to evaluate the number of nodes          *)
(*  exactly L links from a source node in a D-dimensional torus of           *)
(*  width W. When W is odd, this function is defined as                      *)
(*              Reach(L,D,W) =                                               *)
(*                   1   L=0                                                 *)
(*                   2   D=1  and  0<L< = Float(W/2)                         *)
(*                   0   D=1  and  L> Float(W/2)                             *)
(*                                                                           *)
(*                   2* SUM from k=1 to min{L,Float(W/2)}                    *)
(*                        of Reach(L-k,D-1,W)                                *)
(*                  + Reach(L,D-1,W)          Otherwise                      *)
(*                                                                           *)
(*****************************************************************************)

Function Reach (Lr,Dr,Wr: Integer): Real;

Var
 K, Kmax,Wx: Integer;
 Temp: Real;
Label 20;

begin{reach}
  Wx := Wr div 2;
  if Lr = 0 then
  begin{then}
    Reach := 1.0;
    goto 20;
  end;{then}
  if (Dr = 1) and ( (0 < Lr) and (Lr < = Wx)) then
  begin{then}
    if Odd (Wr) then Reach:=2
    else Reach:=1;
    goto 20;
  end;{then}
  if (Dr = 1) and (Lr > Wx) then
  begin{then}
    Reach := 0.0;
    goto 20;
  end;{then}
  if (Lr > = Wx) then Kmax := Wx
  else Kmax := Lr;
  Temp := 0.0;
  for K := 1 to Kmax do
  begin{do}
    if Odd (Wr) then
     Temp := Temp + 2.0*Reach (Lr - k,Dr - 1,Wr)
    else
```

```
    Temp:= Temp + Reach (Lr - k,Dr - 1,Wr);
  end;{do}
  Reach := Temp + Reach (Lr,Dr - 1,Wr);
20: end;{reach}
```

```
(*****************************************************************************)
(*                                                                           *)
(*                    Sphere Procedure                                       *)
(*                                                                           *)
(* This procedure is used to evaluate the mean number of hops                *)
(* when the traffic pattern is Sphere of Locality.                           *)
(* In such a traffic pattern, messages are sent to nodes within              *)
(* a sphere of radius L with probability Probs. Nodes outside                *)
(* the sphere receive messages with a probability (1 - Probs).               *)
(* This traffic pattern is currently implemented for W**D torus              *)
(* only.                                                                     *)
(* Note that we assume that a source node does not send messages             *)
(* to itself.                                                                *)
(*                                                                           *)
(*****************************************************************************)


Procedure Sphere (Ls,Ds,Ws:Integer;Probs:Real);

Var
   Temps1,Temps2,Temps3: Real;
   Ks,Kmax,Network_Diameter:Integer;

begin{sphere}
  Network_Diameter := Ds * ( Ws div 2 );
  if Ls > = Network_Diameter Then Kmax := Network_Diameter
  else Kmax := Ls;
  Temps1 := 0.0;
  Temps2 := 0.0;
  for Ks := 1 to Kmax do   { summation starts from 1, see note above}
  begin{do}
    Temps3 := Reach (Ks,Ds,Ws);
    Temps1 := Temps1 + Ks * Temps3;
    Temps2 := Temps2 + Temps3;
  end;{do}
  N_hops := Probs * Temps1/Temps2;
  if Kmax < Network_Diameter then
  begin{then}
    Temps1 := 0.0;
    Temps2 := 0.0;
    for Ks := Kmax + 1 to Network_Diameter do
    begin{do}
      Temps3 := Reach (Ks,Ds,Ws);
      Temps1 := Temps1 + Ks * Temps3;
      Temps2 := Temps2 + Temps3;
    end;{do}
    N_hops := N_hops + (1.0 - Probs) * Temps1/Temps2;
  end;{then}
end;{sphere}
```

```
(*****************************************************************************)
(*                                                                           *)
(*              Initialize Network Parameters Procedure                      *)
(*                                                                           *)
(* Network parameters are initialized to their default values.               *)
(*                                                                           *)
(*****************************************************************************)


Procedure Initialize_Network_Parameters;

begin{Initialize_Network}
  User_Choice := 0;
  Topology_Index := 1;
  Traffic_Index := 1;
  Switching_Index := 1;
  Band_Width := 10.;
  Message_Length := 512;
  Header_length := 26;
  Processing_Time := 0.1;
  Network_Width := 2;
  Network_Dimension := 10;
  Network_Size := 1024;
{  Beta := 6.0;}
{  Gamma := 0.5; }
 { N_hops := 5.;}
  Curve_Index := 0;
  Curve_Max_Reached := 'No';

  Topology_Type [1] := 'Topology (Binary Torus)';
  Topology_Type [2] := 'Topology (W**D Torus)';
  Topology_Type [3] := 'Topology (W**D Hypercube)';
  Topology_Type [4] := 'Topology (Others)';

  Traffic_Pattern [1] := 'Traffic Pattern (Uniform)';
  Traffic_Pattern [2] := 'Tr. Pat. Locality (torus)';
  Traffic_Pattern [3] := 'Tr. Pat. Locality (Others)';

  Switching_Technique [1] := 'Switching Technique (MS)';
  Switching_Technique [2] := 'Switching Technique (VCT)';

end;{Initialize_Network}
```

```
(*****************************************************************************)
(*                                                                           *)
(*              Display Main Menu Procedure                                  *)
(*                                                                           *)
(* Clear screen, Display main menu, Prompt user to enter choice              *)
(*                                                                           *)
(*****************************************************************************)


Procedure Display_Main_Menue;

begin{Main_Menue}
  ClrScr;
  writeln;
  writeln(' Network Parameters (Current Values)      N=',
                          Network_Size:10,'  Nodes');
  gotoxy (4,4);
  writeln('(1) ',Topology_Type[Topology_Index]);
  gotoxy (43,4);
  writeln ('(2) ', Traffic_Pattern[Traffic_Index]);
  gotoxy (4,5);
  writeln('(3) ',Switching_Technique[Switching_Index]);
  gotoxy (43,5);
  writeln ('(4) Link Bandwidth (Mega Hz)',Band_Width:7:1);
  gotoxy (4,6);
  writeln('(5) Message Length (Bytes)',Message_Length:8);
  gotoxy (43,6);
  writeln ('(6) Header Length (Bytes)',Header_Length:8);
  gotoxy (4,7);
  writeln('(7) Prcocessing Time (m. sec.)',Processing_Time:5:2);
  gotoxy (43,7);
  writeln ('(8) Compute');
  gotoxy (4,8);
  writeln('(9) Plot',Blank:31,'(10) Exit');
  writeln;
  write  (' Enter Your Choice (1..10)          :');

  readln (User_Choice);
end;{Display_Main_Menue}
```

```
(*****************************************************************************)
(*                                                                           *)
(*                    Select Topology Procedure                              *)
(*                                                                           *)
(* Display topology options, Prompt user to enter his choice,                *)
(* Update Network Size.                                                      *)
(*                                                                           *)
(*****************************************************************************)


Procedure Select_Topology;

begin{select_topology}
    ClrScr;
    Writeln (' Topology Options Are:');
    writeln (Blank:10,'(1) Binary Torus' );
    writeln (Blank:10,'(2) W**D Torus ' );
    writeln (Blank:10,'(3) W**D Spanning Bus Hypercube' );
    writeln (Blank:10,'(4) Others' );
    write  (' Enter Your Choice (1..4)  :');
    readln ( Topology_Index );
    case Topology_Index of
    0:  ;
    1: begin{1}
       write (' Enter Network Dimension (Integer Value)  :');
       readln ( Network_Dimension);
       Network_Size :=  exp(Ln(2.0)*Network_Dimension);{2**D}
       end;{1}
   2,3: begin{2,3}
       write (' Enter Network Width (W)  Integer Value  :');
       readln ( Network_Width );
       write  (' Enter Network Dimension (D)  Integer Value  :');
       readln ( Network_Dimension );
       Network_Size :=  exp(In(Network_Width)*Network_Dimension) ;{W**D}
       end;{2,3}
    4: begin{4}
       writeln ('User Provides the Following Parameters',
                              ' Before Computation Starts:');
       writeln ('  N_hops =  Average Number of Hops');
       writeln ('  Beta  =  Communication Processor Loading Factor');
       writeln ('  Gamma  =  Link Traffic Loading Factor');
       writeln;
       write (' Enter Network Size (N) Integer Value  :');
       readln (Network_Size );
       end;{4}
    end;{case}

end;{select topology}
```

```
(*****************************************************************************)
(*                                                                           *)
(*              SELECT Traffic Pattern Procedure                             *)
(*                                                                           *)
(* Clear screen, Display traffic pattern options, and Prompt                 *)
(* user for a choice                                                         *)
(*                                                                           *)
(*****************************************************************************)


Procedure Select_Traffic_Pattern;

Var  choice:integer;
Label  50,60;

begin{select traffic pattern}
    ClrScr;
    writeln (' Traffic Pattern Options Are:');
    writeln (Blank:10,'(1) Uniform Traffic' );
    writeln (Blank:10,'(2) Sphere of Locality (Torus)');
    writeln (blank:10,'(3) Sphere of Locality (Others)');
    writeln;
    write (' Enter Your Choice (1..3)    :');
    read ( Choice );
    case  Choice of
  3: begin{3}
       Traffic_Index := 3;
       writeln;
       writeln ('User Provides the Following Parameters ',
                                  'Before Computation Starts:');
       writeln ('   N_hops =  Average Number of Hops');
       writeln ('   Beta   =  Communication Processor Loading Factor');
       writeln ('   Gamma  =  Link Traffic Loading Factor');
       Delay(5000);
    end;{3}
  2: begin{2}
       Traffic_Index := 2;
{temp} goto 50;
      if  Topology_Index  < > 1 then
      begin{then}
        writeln;
        writeln(' Topology Has Been Changed To Binary Torus                ');
        writeln;
        write ('   Enter Network_Dimension (D)  Integer Value:');
        ReadIn(Network_Dimension);
        Network_Size := exp(ln(2.0) * Network_Dimension);
        Topology_Index := 1;
      end;{ then}
      if (Topology_Index = 1) or (Topology_Index = 2) then
      begin{outer then}
   50: writeln;
```

```
      write (' Enter Radius of Sphere (in Links) :');
      readln (L);
      Network_Diameter := Network_Dimension * ( Network_Width div 2 );
     { goto 60;}{temp}{you should compare to network diameter = d*(W div 2)}
      if L > Network_Diameter then
          begin{then}
            writeln(' This Value is Unlikely    L > Network Diameter ');
            goto 50;
          end;{then}
   60: write (' Enter Probability of Message Within Sphere :');
      readln (Prob);
      if Prob > 1.0 then
          begin{then}
            writeln(' This Value is Unlikely     Probability > 1.0');
            goto 60;
          end;{then}
      end;{outer then}
    end;{2}
  end;{case}
end;{select traffic pattern}
```

```
(*****************************************************************************)
(*                                                                           *)
(*            Select Switching Technique Procedure                           *)
(*                                                                           *)
(* Clear screen, Display switching technique options, and Prompt             *)
(* user to enter choice.                                                     *)
(*                                                                           *)
(*****************************************************************************)


Procedure Select_Switching_Technique;

Var Choice :char;

begin{Select_Switching_Technique}
    ClrScr;
    writeln;
    writeln (' Switching Technique Options are:');
    writeln ('                (1) Message Switching');
    writeln ('                (2) Virtual- Cut-Through');
    writeln;
    write   (' Enter Your Choice (1..2)       :');
    read  (Choice);
    case Choice of
      '0':  ;
      '1': Switching_Index := 1;
      '2': Switching_Index := 2
    end;{case}
end;{Select_Switching_Technique}
```

```
(*****************************************************************************)
(*                                                                           *)
(*                  Select Bandwidth Procedure                               *)
(*                                                                           *)
(*****************************************************************************)

Procedure Select_Link_Bandwidth;

begin{select bandwidth}
  ClrScr;
  writeln;
  writeln (' Current Link Bandwidth is :',Band_Width:7:1,' Mega Hz');
  gotoxy (2,5);
  write ('Enter New BandWidth = ');
  readln (Band_Width);
end;{selct bandwidth}
```

```
(*****************************************************************************)
(*                                                                           *)
(*                  Select Message Length Procedure                          *)
(*                                                                           *)
(*****************************************************************************)

Procedure Select_Message_Length;

begin{selct message length}
  ClrScr;
  writeln;
  writeln (' Current Message Length is :',Message_Length:8,' Bytes');
  gotoxy(2,5);
  write ('Enter New Message Length  :');
  readln (Message_Length);
end;{select message length}
```

```
(*****************************************************************************)
(*                                                                           *)
(*                  Select Header Length Procedure                           *)
(*                                                                           *)
(*****************************************************************************)

Procedure Select_Header_Length;

begin{select header length}
  Clrscr;
  writeln;
  writeln (' Current Header Length is :',Header_Length:8,' Bytes');
  gotoxy (2,5);
  write ('Enter New Header Length   :');
  readln (Header_Length);
end;{select header length}
```

```
(*****************************************************************************)
(*                                                                           *)
(*                  Select processing Time Procedure                         *)
(*                                                                           *)
(*****************************************************************************)

Procedure Select_Processing_Time;

begin{select processing time}
  ClrScr;
  writeln;
  writeln (' Current Processing Time is :',Processing_Time:5:2,' m. sec.');
  gotoxy (2,5);
  write ('Enter New Processing Time   :');
  readln (Processing_Time);
end;{select processing time}
```

```
(*****************************************************************************)
(*                                                                           *)
(*              Compute Message Delay Procedure                              *)
(*                                                                           *)
(* This procedure computes end-to-end communication delay time of            *)
(* message assuming either message switching or virtual-cut-through          *)
(* switching technique. Delay time is studied versus packet                  *)
(* generation rate. The following global parameters are used:                *)
(*       Message_Length, Header_Length, Processing_Time,                     *)
(*       Band_Width, Beta, Gamma,N_Hops, Num_Points, Step.                   *)
(*                                                                           *)
(*****************************************************************************)


Procedure Compute_Message_Delay;

Var
  I: Integer;
  Rho, Alpha: Real;
  Local_Step, Mu1, Mu2, Q1, Q2: Real;
  Delay_CP, Delay_L: Real;
  Lamda, Lamda_CP, Lamda_L: Real;

  Lamda_FP : Array[1..Num_Point] of Real;
  Delay_MS : Real;
  Delay_VCT : Real;

Label 20, 30, 40;

begin{Compute_Message_Dlay}
     Curve_Index := Curve_Index + 1;
     Legend[Curve_Index] := Curve_Index;

     Mu1 := 1000. / Processing_Time;   {CP_Service_Rate in Hz}
     Mu2 := (Band_Width * (1000000.0)) / (Message_Length * 8. );
                                {Link_Service_Rate in Hz}
     Local_Step := Step;
     Alpha := Header_Length / Message_Length;

     Lamda_FP [1] := 0.01 - Step;
     for I := 1 to Num_Point do
     begin{for}
   30: Lamda := Lamda_FP[I] + Local_Step;
       Lamda_CP := Beta * Lamda;
       Lamda_L  := Gamma * Lamda;
       Q1 := 2. * Mu1 * ( Mu1 - Lamda_CP );
       Q2 := Mu2 - Lamda_L;

       if (( Q1 < = 0.0)  or ( Q2 < = 0.0 ) )  then
   20:      begin{then}
            Local_Step := Local_Step /20.;
```

```
          goto 30;
        end;{then}
      Delay_CP := 1000. * ( 1. / Mu1  + Lamda_CP / Q1 ); {time in m. sec.}
      Delay_L := 1000. / Q2;                          {time in m.sec.}
      Delay_MS  := (N_hops + 1.0) * Delay_CP + N_hops * Delay_L;
      if Delay_MS > Delay_Max  then  goto 20;
  40: Rho := Lamda / Mu2;          {Link Utilization Factor}
      Delay_VCT  := Delay_MS  - (N_hops - 1.0) * (1.0 - Rho) *
                        (Delay_CP  +  1000.0*(1.0 - Alpha) / Mu2);
      Lamda_FP[I + 1] := Lamda;
      Lamda_FP[I] := Lamda;
      Xglobal[Curve_Index,I] := Lamda;
      if Switching_Index = 1 then Yglobal[Curve_Index,I] := Delay_MS;
      if Switching_Index = 2 then Yglobal[Curve_Index,I] := Delay_VCT;
    end;{for}
    writeln;
    writeln(' This is Curve # ',Curve_Index);
    delay(2000);
end;{Compute_Message_Delay}
```

```
(*****************************************************************************)
(*                                                                           *)
(*              Evaluate Network Parameters Procedure                        *)
(*                                                                           *)
(* This procedure is called before each computation.                         *)
(* Note#1                                                                    *)
(*      N_hops is scaled to exclude nodes routing messages to                *)
(*      themselves.                                                          *)
(*                                                                           *)
(*****************************************************************************)


Procedure Evaluate_Network_Parameters;

Var Ans: Char;
Label 80, 90;

begin{evaluation}
  if Curve_Index = Curve_Max then
  begin{then}
    writeln (' Maximum Number of Curves Has Been Reached                ');
    writeln (' New Curves will Replace Oldest Curves');
    Delay(5000);
    Curve_Index := 0;
    Curve_Max_Reached := 'yes';
  end;{then}
  case Traffic_Index of
  1: begin{1}
    if Topology_Index = 1 then
    begin{1,1}                                    {Note#1}
      N_hops := (Network_Dimension*Network_Size) / (2.0*(Network_Size -1.0));
      Beta := N_hops + 1.0;
      Gamma := N_hops / Network_Dimension;
    end;{1,1}
    if Topology_Index = 2 then
    begin{1,2}
      if Odd(Network_Width) then
      N_hops := (Network_width*Network_Width - 1.0)*
                ( Network_Dimension/(4.0* Network_Width) )*
                (Network_Size / (Network_Size -1.0 )) {Note#1}
      else
      N_hops := (Network_Width*Network_Dimension/4.0)*
                (Network_Size/(Network_Size - 1));   {Note#1}
      Beta := N_hops + 1.0;
      Gamma := N_hops/Network_Dimension;
    end;{1,2}
    if Topology_Index = 3 then
    begin{1,3}
      N_hops := (Network_Width - 1.0)*Network_Dimension/Network_Width;
      Beta := N_hops + 1.0;
      Gamma := Network_Width - 1.0;
```

```
    end;{1,3}
    if Topology_Index = 4 then goto 80;
    end;{1}
  2: begin{2}
     Sphere (L,Network_Dimension,Network_Width,Prob);
     Beta := N_hops + 1.0;
     Gamma := N_hops/Network_Dimension;
    end;{2}
  3: begin{3}
80:   writeln (' User Has To Provide N_hops, Beta, Gamma ');
     writeln;
     write (' Do You Have These Values Ready   (Y/N) ?');
     readln (Ans);
     if (Ans = 'y') or (Ans = 'Y') then
     begin{then}
       write (' Enter Average Number Of Hops  N_hops =');
       readln (N_hops);
       write (' Enter Processor Traffic Load Factor  Beta=');
       readln ( Beta );
       write (' Enter Link Traffic Load Factor  Gamma=');
       readln ( Gamma );
     end;{then}
     if (Ans = 'n') or (Ans = 'N') then goto 90;
    end;{3}
  end;{case}
  Compute_Message_Delay;
90: end;{evaluation}
```

```
(*****************************************************************************)
(*                                                                           *)
(*                  Select Plot Curves Procedure                             *)
(*                                                                           *)
(* This procedure is called before plotting the curves.                      *)
(* It uses Curve_Index and Arrays Xglobal, Yglobal to get                    *)
(* Temp_Curve_index, Arrays Xtemp, Ytemp that will be used by                *)
(* Procedure Plot.                                                           *)
(*                                                                           *)
(*****************************************************************************)


Procedure Select_Plot_Curves;

Var
  I,J,Kp: Integer;
  Ans: Char;

Label 20;

begin{select plot curve}
  if Curve_Index = 0 then
  begin{then}
     writeln(' No Data Yet To Plot                                          ');
     Sound(440);
     Delay(50);
     NoSound;
     Delay(3000);
     goto 20;
  end;{then}
    if Curve_Max_Reached = 'yes' then Kp := Curve_Max
    else Kp := Curve_Index;
    Temp_Curve_Index := Kp;
  write (' do you want to plot all curves  (Y/N)   ?');
  readln ( Ans );
  if (Ans = 'y')  or ( Ans  = 'Y') or (Ans = 'yes') then
  begin{then}
    for j := 1 to Kp do
    begin{outer do}
      for I := 1 to Num_Point do
      begin{inner do}
        Xtemp[J,I] := Xglobal[J,I];
        Ytemp[J,I] := Yglobal[J,I];
        Legend[J] := J;
      end;{inner do}
    end;{outer do}
    goto 20;
  end;{then}

  Temp_Curve_Index := 0;
  for J:=1 to Kp do
```

```
 begin{outer do}
   write (' Do You Want To Plot Curve #',J:3,'  (Y/N) ?');
   readln (Ans);
   if Ans ='y' then
   begin{then}
     Temp_Curve_Index := Temp_Curve_Index + 1;
     for I := 1 to Num_Point do
     begin{inner do}
       Xtemp[Temp_curve_Index,I] := Xglobal[J,I];
       Ytemp[Temp_curve_Index,I] := Yglobal[J,I];
     end;{inner do}
     Legend[Temp_Curve_Index] := J;
   end;{then}
 end;{outer do}
20: end;{select plot curves}
```

```
(*****************************************************************************)
(*                                                                           *)
(*                          Plot Procedure                                   *)
(*                                                                           *)
(*****************************************************************************)


Procedure plot;

var
  xscale,yscale: Real;
  x1,x2,y1,y2: Integer;
  temp: Real;
  I,J: Integer;
  Continue: Char;
Label 70;

{$I graph.p }
begin{plot}
  if Curve_Index = 0  then goto 70;
  Xaxl := 360;
  Yaxl := 120;
{temp}  write (' Enter X_Max   (Integer Value)     :');
  Readln(X_Max);
  Y_Max := 50;{end temp}
  write (' Enter Y_Max   (Integer Value)     :');
  Readln (Y_Max );
  xdiv := X_Max div 5;
  ydiv := 10;
  Title:= ' Delay Time Versus Packet Generation Rate';

            {Screen Set Up}

  TextMode(2);
  HiRes;
  HiResColor(White);

              {Write Scales,Title}

  gotoxy(1,19);
  writeln('      0',xdiv:10,2*xdiv:9,3*xdiv:9,4*xdiv:9,5*xdiv:9);
  writeln( '                Lambda (Packet/Sec.)     ');
  writeln('       ',Title);

  gotoxy (1,4);
  Writeln(' D ');
  writeln(' e ');
  Writeln(' l ');
  writeln(' a ');
  writeln(' y ');
  writeln;
```

```
writeln(' i ');       {Write Y_Axis Label}
writeln(' n ');
writeln;
writeln(' m ');
writeln;
writeln(' s ');
writeln(' e ');
writeln(' c ');

gotoxy (3,3);
ydiv := Y_Max div 5;
write(5*ydiv:4);
gotoxy (4,6);
write(4*ydiv:4);
gotoxy(4,9);
write(3*ydiv:4);      {Write Y_Axis Scale}
gotoxy (4,12);
write(2*ydiv:4);
gotoxy (4,15);
write(ydiv:4);
gotoxy (4,18);
write('  0');


gotoxy (1,23);
write ('Enter Carriage Return To Continue    Waiting                    ');
  Sound(440);
  Delay(50);
  NoSound;

                {write curve legends}

xscale := 45 / X_Max ;
yscale := 15 / Y_Max ;
for J := 1 to Temp_Curve_Index do
begin{big do}
 for I := 1 to Num_Point do
 begin{small do}
   x[I] := Xtemp[J,I];
   y[I] := Ytemp[J,I];
 end;{small do}

 I := 15;{ ( Num_Point - 20 );}
 x[I] := Xtemp[J,I+J];
 y[I] := Ytemp[J,I+J];
          {scale legend points}
 x1 := 7 + Round((x[I] * xscale));
 y1 := 18 - Round((y[I] * yscale));
 gotoxy (x1,y1);
 writeln ('#',Legend[J]);
end;{big do}
```

```
          {set Screen for Graphics}
Palette(1);
ColorTable (0,1,2,3);

          {Draw Borders}
GraphWindow (60,15,Xaxl+60,Yaxl+15);
Draw(0,0,0,Yaxl,1);
Draw(0,Yaxl,Xaxl,Yaxl,1);
Draw(Xaxl,Yaxl,Xaxl,0,1);
Draw(Xaxl,0,0,0,1);

            {Draw Tic Marks}
xtic := Xaxl div 5;
ytic := Yaxl div 5;
for  I:= 1 to 4  do
begin {do}
  Draw (I*xtic,Yaxl,I*xtic,Yaxl-1,1);
  Draw (0,I*ytic,2,I*ytic,1);
end;{do}

            {Plot Curves}

xscale := Xaxl / X_Max ;
yscale := Yaxl / Y_Max ;
for J := 1 to Temp_Curve_Index do
begin{big do}
  for I := 1 to Num_Point do
  begin{small do}
    x[I] := Xtemp[J,I];
    y[I] := Ytemp[J,I];
  end;{small do}

        {scale curve points and Plot Them}

   for I := 1 to ( Num_Point - 1 ) do
   begin{small do}
   x1 := Round(x[I] * xscale);
   x2 := Round(x[I+1] * xscale);
   y1 := Yaxl -  Round(y[I] * yscale);
   y2 := Yaxl -  Round(y[I+1] * yscale);
   Draw (x1,y1,x2,y2,1);
 end;{small do}
end;{big do}
Read ( Continue );

              {Back to Text Mode}
Textmode;
```

```
70:  end;{Plot}
(*****************************************************************************)
(*                                                                           *)
(*                        Write Data File Procedure                          *)
(*                                                                           *)
(*****************************************************************************)


Procedure Write_Data_File;
Var
   J,I: Integer;
   FileName: string [15];

begin{write data file}
  write( ' Writing a Data File .. Enter Data File Name:  ');
  Readln (FileName);
  Assign (Fil,FileName);{'Network.Dat');}
  Rewrite ( Fil );
  for J := 1  to Curve_Index do
  begin{big do}
    for I := 1 to Num_Point do
    writeln (Fil,Xglobal[J,I]:15:5,Yglobal[J,I]:15:5);
  end;{big do}
  Close (Fil);
  writeln;
  writeln ('  Data File Name is:   ',  FileName);{Network.Dat ')};
  writeln ('  Data Format is (2F15.5)');
  writeln ('  Number of Points is',Num_Point:5,'  For Each Curve');
end;{write data file}
```

```
(*****************************************************************************)
(*                                                                           *)
(*                          Main Program                                     *)
(*                                                                           *)
(*****************************************************************************)


begin{program}
  Initialize_Network_Parameters;
  while User_Choice < 10 do
  begin{while}
    Display_Main_Menue;
    case User_Choice of
      1: Select_Topology;
      2: Select_Traffic_Pattern;
      3: Select_Switching_Technique;
      4: Select_Link_Bandwidth;
      5: Select_Message_Length;
      6: Select_Header_Length;
      7: Select_Processing_Time;
      8: Evaluate_Network_Parameters;{ and Compute_Message_Delay}
      9: begin{9}
           Select_Plot_Curves;
           Plot;
         end;{9}
    end;{case}
  end;{while}
  Write_Data_File;
  writeln ( N_hops:5);
end.{program}
```

Appendix B. THE M/M/1 QUEUE

The following presents some of the important properties of the M/M/1 queueing system as outlined in [25]. The M/M/1 is characterized by having a Poisson input and a single server with an exponential service time. A Poisson input means that interarrival times follow an exponential probability distribution. The server utilization factor, ρ, is defined as the average arrival rate of customers to the system times the average service time each requires. If λ and μ are used to denote the mean arrival rate at the queue input and the mean service rate, respectively, then ρ is given by [25]

$$\rho = \frac{\lambda}{\mu}$$

The following relations hold for an M/M/1 queueing system when customers are served on first-come-first-serve basis [25]:

$$\text{average waiting time} = \frac{\rho/\mu}{1-\rho}$$

$$\text{average service time} = \frac{1}{\mu}$$

$$\text{total average delay} = \text{average service time} + \text{average waiting time}$$

$$= \frac{1}{\mu} + \frac{\rho/\mu}{1-\rho}$$

$$= \frac{1/\mu}{1-\rho} = \frac{1}{\mu - \lambda}$$

$$\text{probability of the server being idle} = (1 - \rho)$$

The vita has been removed
from the scanned document